Ben Goertzel with Cassio Pennachin & Nil Geisweiller & the OpenCog Team

Building Better Minds:

Artificial General Intelligence via the CogPrime Architecture

September 3, 2012

Preface

Please note, you are now reading a preliminary draft of this book. It's basically complete as regards contents, but still needs some editing, and some formatting fixes. And it hasn't been checked thoroughly yet so some errors may remain. Reader beware.

This is a large book with an even larger goal: is to outline a practical approach to creating software systems with general intelligence at the human level and ultimately beyond.

Part I reviews various critical conceptual issues, then outlines the broad sketches of the CogPrime

design for an AGI (Artificial General Intelligence) system and a method for giving a young AGI system appropriate experience. Along the way a formal theory of general intelligence is sketched, and a broad roadmap from here to human-level artificial intelligence. Hints are also given regarding how to eventually, potentially advance even beyond human level, including some speculations about strongly self-modifying AGI architectures with flexibility far exceeding that of the human brain.

Part II then digs into the details of CogPrime

's multiple structures, processes and functions, culminating in an argument as to why CogPrime

will be able to achieve human-level AGI, and a discussion of how a CogPrime

-powered virtual agent or robot would handle some simple practical tasks such as social play with blocks in a preschool context. In it, we first describe the CogPrime

software architecture and knowledge representation in detail; then review the cognitive cycle via which CogPrime

perceives and acts in the world and reflects on itself. We then turn to various forms of learning: procedural, declarative (e.g. inference), simulative and integrative. Methods of enabling natural language functionality in CogPrime

are then discussed; and then the volume concludes with a chapter summarizing the argument that CogPrime

can lead to human-level (and eventually perhaps greater) AGI, and a chapter giving a thought experiment describing the internal dynamics via which a completed CogPrime

system might solve the problem of obeying the request "Build me something with blocks that I haven't seen before."

The chapters here are written to be read in linear order; but the impatient reader might wish to start with the first few chapters, then skim the final two chapters, and then return to reading in linear order. The final two chapters give a broad overview of why we think the CogPrime

design will work, in a way that depends on the technical details of the previous chapters, but (perhaps) not so sensitively as to be incomprehensible without them.

This is admittedly an unusual sort of book, mixing demonstrated conclusions with unproved conjectures in a complex way, and aiming at an extraordinarily ambitious goal. However, it is not mere armchair speculation – the ideas described here are currently being used as the basis for an open-source software project called OpenCog, which is being worked on by software developers around the world. Right now OpenCog embodies only a percentage of the overall CogPrime

design. However, if OpenCog continues to attract sufficient funding or volunteer interest, then the ideas presented in these volumes. will be validated or refuted via practice.

We recall the words of Sir Edmund Hillary, who first scaled Everest, and wrote a book titled: "Nothing Venture, Nothing Win."

The writing of this book began in earnest in 2001, at which point it was informally referred to as "The Novamente Book." The original "Novamente Book" manuscript ultimately got too big for its own britches, and subdivided into a number of different works – *The Hidden Pattern* [Goe06a], a philosophy of mind book published in 2006; *Probabilistic Logic Networks* [GIGH08], a more technical work published in 2008; *Real World Reasoning* [GCG+11], a sequel to *Probabilistic Logic Networks* published in 2011; and the two parts of this book.

The ideas described in this book have been the collaborative creation of multiple overlapping communities of people over a long period of time. The core concepts of the CogPrime

design and the underlying theory were conceived by Ben Goertzel in the period 1995-1996 when he was a Research Fellow at the University of Western Australia; but those early ideas have been elaborated and improved by many more people than can be listed here (as well as by Ben's ongoing thinking and research). The collaborative design process ultimately resulting in CogPrime

started in 1997 when Intelligenesis Corp. was formed – the Webmind AI Engine created in Intelligenesis's research group during 1997-2001 was the predecessor to the Novamente Cognition Engine created at Novamente LLC during 2001-2008, which was the predecessor to CogPrime

. Many of the chapters here have co-authors beyond the three co-authors of the book, but the set of chapter co-authors does not exhaust the set of significant contributors to the ideas presented.

This is a draft version; appropriate acknowledgements will be inserted here when we finalize the book! As a temporary stopgap we wish to express our profound gratitude to the entire multi-multi-...-multiverse.

July 2012
Ben Goertzel
Cassio Pennachin
Nil Geisweiller

Contents

Part 1
A Path to Beneficial Artificial General Intelligence

Chapter 1
Introduction

1.1 AI Returns to Its Roots

Our goal in this book is straightforward: to present a conceptual and technical design for a thinking machine, a software program capable of the same qualitative sort of general intelligence as human beings. We don't know exactly how far the design outlined here will be able to take us, but it seems plausible that once fully implemented, tuned and tested, it will be able to achieve general intelligence at the human level and in some respects perhaps beyond.

We don't view the design presented here, CogPrime , as the unique path to advanced artificial general intelligence (AGI), nor as the ultimate end-all of AGI research. We feel confident there are multiple possible paths to advanced AGI, and that in following any of these paths, multiple theoretical and practical lessons will be learned, leading to modifications of the ideas possessed while along the early stages of the path. But our goal here is to articulate **one** path that we believe makes sense to follow, one overall design that we believe can work.

An outsider to the AI field might think this sort of book commonplace in the research literature, but insiders know that's far from the truth. The field of Artificial Intelligence (AI) was founded in the mid 1950s with the aim of constructing "thinking machines" - that is, computer systems with human-like general intelligence, including humanoid robots that not only look but act and think with intelligence equal to and ultimately greater than human beings. But in the intervening years, the field has drifted far from its ambitious roots, and this book represents part of a movement aimed at restoring the initial goals of the AI field, but in a manner powered by new tools and new ideas far beyond those available half a century ago.

After the first generation of AI researchers found the task of creating human-level AGI very difficult given the technology of their time, the AI field shifted focus toward what Ray Kurzweil has called "narrow AI" – the

understanding of particular specialized aspects of intelligence; and the creation AI systems displaying intelligence regarding specific tasks in relatively narrow domains. In recent years, however, the situation has been changing. More and more researchers have recognized the necessity – and feasibility – of returning to the original goals of the field.

In the decades since the 1950s, cognitive science and neuroscience have taught us a lot about what a cognitive architecture needs to look like to support roughly human-like general intelligence. Computer hardware has advanced to the point where we can build distributed systems containing large amounts of RAM and large numbers of processors, carrying out complex tasks in real time. The AI field has spawned a host of ingenious algorithms and data structures, which have been successfully deployed for a huge variety of purposes.

Due to all this progress, increasingly, there has been a call for a transition from the current focus on highly specialized "narrow AI" problem solving systems, back to confronting the more difficult issues of "human level intelligence" and more broadly "artificial general intelligence (AGI)." Recent years have seen a growing number of special sessions, workshops and conferences devoted specifically to AGI, including the annual BICA (Biologically Inspired Cognitive Architectures) AAAI Symposium, and the international AGI conference series (AGI-08 , AGI-09, AGI-10, AGI-11). And, even more exciting, as reviewed in Chapter 4, there are a number of contemporary projects focused directly and explicitly on AGI (sometimes under the name "AGI", sometimes using related terms such as "Human Level Intelligence").

In spite of all this progress, however, no one has yet clearly articulated a detailed, systematic design for an AGI, with potential to yield general intelligence at the human level and ultimately beyond. Perhaps the most comprehensive attempts in this direction have been the works of Stan Franklin and Joscha Bach, to be discussed in Chapter 4; but as we will discuss later on, while we believe both of their approaches are basically conceptually sound, we also feel their designs have shortcomings and lacunae alongside their considerable strengths.

In this spirit, our main goal in this lengthy two-part book is to outline a novel *design for a thinking machine* – an AGI design which we believe has the capability to produce software systems with intelligence at the human adult level and ultimately beyond. Many of the technical details of this design have been previously presented online in a wikibook [**?**]; and the basic ideas of the design have been presented briefly in a series of conference papers [**?**, GPPG06, Goe09b]. But the overall design has not been presented in a coherent and systematic way before this book. In order to frame this design properly, we also present a considerable number of broader theoretical and conceptual ideas here, some more and some less technical in nature.

The AGI design presented here has not previously been granted a name independently of its particular software implementations, but for the purposes of this book it needs a name, so we've christened it **CogPrime** . This fits with

the name "OpenCogPrime " that has already been used to describe the software implementation of CogPrime within the open-source OpenCog AGI software framework. The OpenCogPrime software, right now, implements only a small fraction of the CogPrime design as described here. However, OpenCog was designed specifically to enable efficient, scalable implementation of the full CogPrime design (as well as to serve as a more general framework for AGI R&D); and work currently proceeds in this direction, though there is a lot of work still to be done and many challenges remain. [1]

The CogPrime design is more comprehensive and thorough than anything that has been presented in the literature previously, including the work of others reviewed in Chapter 4. It covers all the key aspects of human intelligence, and explains how they interoperate and how they can be implemented in digital computer software. Part 1 of this work outlines CogPrime at a high level, and makes a number of more general points about artificial general intelligence and the path thereto; then Part 2 digs deeply into the technical particulars of CogPrime . Even Part 2, however, doesn't explain all the details of CogPrime that have been worked out so far, and it definitely doesn't explain all the implementation details that have gone into designing and building OpenCogPrime . Creating a thinking machine is a large task, and even the intermediate level of detail takes up a lot of pages.

1.2 The Secret Sauce

There is no consensus on why all the related technological and scientific progress mentioned above has not yet yielded AI software systems with human-like general intelligence. However, we hypothesize that the core reason boils down to the following three points:

- Intelligence depends on the emergence of certain high-level structures and dynamics across a system's whole knowledge base;
- We have not discovered any one algorithm or approach capable of yielding the emergence of these structures;

[1] This brings up a terminological note: At several places in this Volume and the next we will refer to the current CogPrime or OpenCog implementation; in all cases this refers to OpenCog as of mid 2012. We realize the risk of mentioning the state of our software system at time of writing: for future readers this may give the wrong impression, because if our project goes well, more and more of CogPrime will get implemented and tested as time goes on (e.g. within the OpenCog framework, under active development at time of writing). However, not mentioning the current implementation at all seems an even worse course to us, since we feel readers will be interested to know which of our ideas – at time of writing – have been honed via practice and which have not. Online resources such as http://opencog.org may be consulted by readers curious about the current state of the main OpenCog implementation; though in future forks of the code may be created, or other systems may be built using some or all of the ideas in this book, etc.

- Achieving the emergence of these structures within a system formed by integrating a number of different AI algorithms and structures requires careful attention to the manner in which these algorithms and structures are integrated; and so far the integration has not been done in the correct way.

The human brain appears to be an integration of an assemblage of diverse structures and dynamics, built using common components and arranged according to a sensible cognitive architecture. However, its algorithms and structures have been honed by evolution to work closely together – they are very tightly inter-adapted, in the same way that the different organs of the body are adapted to work together. Due their close interoperation they give rise to the overall systemic behaviors that characterize human-like general intelligence. We believe that the main missing ingredient in AI so far is **cognitive synergy**: the fitting-together of different intelligent components into an appropriate cognitive architecture, in such a way that the components richly and dynamically support and assist each other, interrelating very closely in a similar manner to the components of the brain or body and thus giving rise to appropriate emergent structures and dynamics. Which leads us to one of the central hypotheses underlying the CogPrime approach to AGI: that **the cognitive synergy ensuing from integrating multiple symbolic and subsymbolic learning and memory components in an appropriate cognitive architecture and environment, can yield robust intelligence at the human level and ultimately beyond.**

The reason this sort of intimate integration has not yet been explored much is that it's difficult on multiple levels, requiring the design of an architecture and its component algorithms with a view toward the structures and dynamics that will arise in the system once it is coupled with an appropriate environment. Typically, the AI algorithms and structures corresponding to different cognitive functions have been developed based on divergent theoretical principles, by disparate communities of researchers, and have been tuned for effective performance on different tasks in different environments. Making such diverse components work together in a truly synergetic and cooperative way is a tall order, yet we believe that this – rather than some particular algorithm, structure or architectural principle – is the "secret sauce" needed to create human-level AGI based on technologies available today.

1.3 Extraordinary Proof?

There is a saying that "extraordinary claims require extraordinary proof" and by that standard, if one believes that having a design for an advanced AGI is an extraordinary claim, this book must be rated a failure. We don't offer extraordinary proof that CogPrime , once fully implemented and educated, will be capable of human-level general intelligence and more.

It would be nice if we could offer mathematical proof that CogPrime has the potential we think it does, but at the current time mathematics is simply not up to the job. We'll pursue this direction briefly in 7 and other chapters, where we'll clarify exactly what kind of mathematical claim "CogPrime has the potential for human-level intelligence" turns out to be. Once this has been clarified, it will be clear that current mathematical knowledge does not yet let us evaluate, or even fully formalize, this kind of claim. Perhaps one day rigorous and detailed analyses of practical AGI designs will be feasible – and we look forward to that day – but it's not here yet.

Also, it would of course be profoundly exciting if we could offer dramatic practical demonstrations of CogPrime 's capabilities. We do have a partial software implementation, in the OpenCogPrime system, but currently the things OpenCogPrime does are too simple to really serve as proofs of CogPrime 's power for advanced AGI. We have used some CogPrime ideas in the OpenCog framework to do things like natural language understanding and data mining, and to control virtual dogs in online virtual worlds; and this has been very useful work in multiple senses. It has taught us more about the CogPrime design; it has produced some useful software systems; and it constitutes fractional work building toward a full OpenCog based implementation of CogPrime . However, to date, the things OpenCogPrime has done are all things that could have been done in different ways without the CogPrime architecture (though perhaps not as elegantly nor with as much room for interesting expansion).

The bottom line is that building an AGI is a big job. Software companies like Microsoft spend dozens to hundreds of man-years building software products like word processors and operating systems, so it should be no surprise that creating a digital intelligence is also a relatively large-scale software engineering project. As time advances and software tools improve, the number of man-hours required to develop advanced AGI gradually decreases – but right now, as we write these words, it's still a rather big job. In the OpenCogPrime project we are making a serious attempt to create a CogPrime based AGI using an open-source development methodology, with the open-source Linux operating system as one of our inspirations. But the open-source methodology doesn't work magic either, and it remains a large project, currently at an early stage. I emphasize this point so that readers lacking software engineering expertise don't take the currently fairly limited capabilities of OpenCogPrime as somehow damning about the potential of the CogPrime design. The design is one thing, the implementation another – and the OpenCogPrime implementation currently encompasses perhaps one third to one half of the key ideas in this book.

So we don't have extraordinary proof to offer. What we aim to offer instead are clearly-constructed conceptual and technical arguments as to why we think the CogPrime design prime has dramatic AGI potential.

It is also possible to push back a bit on the common intuition that having a design for human-level AGI is such an "extraordinary claim." It may be

extraordinary relative to contemporary science and culture, but we have a strong feeling that the AGI problem is not difficult in the same ways that most people (including most AI researchers) think it is. We suspect that in hindsight, after human-level AGI has been achieved, people will look back in shock that it took humanity so long to come up with a workable AGI design. As you'll understand once you've finished Part 1 of the book, we don't think general intelligence is nearly as "extraordinary" and mysterious as it's commonly made out to be. Yes, building a thinking machine is hard – but humanity has done a lot of other hard things before. It may seem difficult to believe that human-level general intelligence could be achieved by something as simple as a collection of algorithms linked together in an appropriate way and used to control an agent. But we suggest that, once the first powerful AGI systems are produced, it will become clear and apparent engineering human-level minds is not so profoundly different from engineering other complex systems.

All in all, we'll consider the book successful if a significant percentage of open-minded, appropriately-educated readers come away from it scratching their chins and pondering: *"Hmm. You know, that just might work."* É and a small percentage come away thinking *"Now that's an initiative I'd really like to help with!"*.

1.4 Potential Approaches to AGI

In principle, there is a large number of approaches one might take to building an AGI, starting from the knowledge, software and machinery now available. This is not the place to review them in detail, but a brief list seems apropos, including commentary on why these are not the approaches we have chosen for our own research. Our intent here is not to insult or dismiss these other potential approaches, but merely to indicate why, as researchers with limited time and resources, we have made a different choice regarding where to focus our own energies.

Build AGI from Narrow AI

Most of the AI programs around today are "narrow AI" programs – they carry out one particular kind of task intelligently. One could try to make an advanced AGI by combining a bunch of enhanced narrow AI programs inside some kind of overall framework.

However, we're rather skeptical of this approach because none of these narrow AI programs have the ability to generalize across domains – and we don't see how combining them or extending them is going to cause this to magically emerge.

Enhancing Chatbots

One could seek to make an advanced AGI by taking a chatbot, and trying to improve its code to make it actually understand what it's talking about. We have some direct experience with this route, as in 2010 our AI consulting firm was contracted to improve Ray Kurzweil's online chatbot "Ramona." Our new Ramona understands a lot more than the previous Ramona version or a typical chatbot, due to using Wikipedia and other online resources, but still it's far from an AGI.

A more ambitious attempt in this direction was Jason Hutchens' a-i.com project, which sought to create a human child level AGI via development and teaching of a statistical learning based chatbot (rather than the typical rule-based kind). The difficulty with this approach, however, is that the architecture of a chatbot is fundamentally different from the architecture of a generally intelligent mind. Much of what's important about the human mind is not directly observable in conversations, so if you start from conversation and try to work toward an AGI architecture from there, you're likely to miss many critical aspects.

Emulating the Brain

One can approach AGI by trying to figure out how the brain works, using brain imaging and other tools from neuroscience, and then emulating the brain in hardware or software.

One rather substantial problem with this approach is that we don't really understand how the brain works yet, because our software for measuring the brain is still relatively crude. There is no brain scanning method that combines high spatial and temporal accuracy, and none is likely to come about for a decade or two. So to do brain-emulation AGI seriously, one needs to wait a while until brain scanning technology improves.

Current AI methods like neural nets that are loosely based on the brain, are really not brain-like enough to make a serious claim at emulating the brain's approach to general intelligence. We don't yet have any real understanding of how the brain represents abstract knowledge, for example, or how it does reasoning (though the authors, like many others, have made some speculations in this regard [GMIH08]).

Another problem with this approach is that once you're done, what you get is something with a very humanlike mind, and we already have enough of those! However, this is perhaps not such a serious objection, because a digital-computer-based version of a human mind could be studied much more thoroughly than a biology-based human mind. We could observe its dynamics in real-time in perfect precision, and could then learn things that would allow us to build other sorts of digital minds.

Evolve an AGI

Another approach is to try to run an evolutionary process inside the computer, and wait for advanced AGI to evolve.

One problem with this is that we don't know how evolution works all that well. There's a field of artificial life, but so far its results have been fairly disappointing. It's not yet clear how much one can vary on the chemical structures that underly real biology, and still get powerful evolution like we see in real biology. If we need good artificial chemistry to get good artificial biology, then do we need good artificial physics to get good artificial chemistry?

Another problem with this approach, of course, is that it might take a really long time. Evolution took billions of years on Earth, using a massive amount of computational power. To make the evolutionary approach to AGI effective, one would need some radical innovations to the evolutionary process (such as, perhaps, using probabilistic methods like BOA [**?**] or MOSES [**?**] in place of traditional evolution).

Derive an AGI design mathematically

One can try to use the mathematical theory of intelligence to figure out how to make advanced AGI.

This interests us greatly, but there's a huge gap between the rigorous math of intelligence as it exists today and anything of practical value. As we'll discuss in Chapter 7, most of the rigorous math of intelligence right now is about how to make AI on computers with dramatically unrealistic amounts of memory or processing power. When one tries to create a theoretical understanding of real-world general intelligence, one arrives at quite different sorts of considerations, as we will roughly outline in Chapter 10. Ideally we would like to be able to study the CogPrime design using a rigorous mathematical theory of real-world general intelligence, but at the moment that's not realistic. The best we can do is to conceptually analyze CogPrime and its various components in terms of relevant mathematical and theoretical ideas; and perform analysis of CogPrime 's individual structures and components at varying levels of rigor.

Use heuristic computer science methods

The computer science field contains a number of abstract formalisms, algorithms and structures that have relevance beyond specific narrow AI applications, yet aren't necessarily understood as thoroughly as would be required to integrate them into the rigorous mathematical theory of intelligence. Based on these formalisms, algorithms and structures, a number of "single formalis-

m/algorithm focused" AGI approaches have been outlined, some of which will be reviewed in Chapter 4. For example Pei Wang's NARS approach is based on a specific logic which he argues to be the "logic of general intelligence" – so, while his system contains many other aspects than this logic, he considers this logic to be the crux of the system and the source of its potential power as a AGI system.

The basic intuition on the part of these "single formalism/algorithm focused" researchers seems to be that there is one key formalism or algorithm underlying intelligence, and if you achieve this key aspect in your AGI program, you're going to get something that fundamentally thinks like a person, even if it has some differences due to its different implementation and embodiment. On the other hand, it's also possible that this idea is philosophically incorrect: that there is no one key formalism, algorithm, structure or idea underlying general intelligence. The CogPrime approach is based on the intuition that to achieve human-level, roughly human-like general intelligence based on feasible computational resources, one needs an appropriate heterogeneous combination of algorithms and structures, each coping with different types of knowledge and different aspects of the problem of achieving goals in complex environments.

Integrative Cognitive Architecture

Finally, to create advanced AGI one can try to build some sort of integrative cognitive architecture: a software system with multiple components that each carry out some cognitive function, and that connect together in a specific way to try to yield overall intelligence.

Cognitive science gives us some guidance about the overall architecture, and computer science and neuroscience give us a lot of ideas about what to put in the different components. But still this approach is very complex and there is a lot of need for creative invention.

This is the approach we consider most "serious" at present (at least until neuroscience advances further) – it's the one underlying Franklin's and Bach's approaches, of which we've spoken admiringly above. And, as will be discussed in depth in these pages, this is the approach we've chosen: CogPrime is an integrative AGI architecture.

1.4.1 Can Digital Computers Really Be Intelligent?

All the AGI approaches we've just mentioned assume that it's possible to make AGI on digital computers. While we suspect this is correct, we must note that it isn't proven.

It might be that – as Penrose [Pen96], Hameroff [Ham87] and others have argued – we need quantum computers or quantum gravity computers to make AGI. However, there is no evidence of this at this stage. Of course the brain like all matter is described by quantum mechanics, but this doesn't imply that the brain is a "macrosopic quantum system" in a strong sense (like, say, a Bose-Einstein condensate). And even if the brain does use quantum phenomena in a dramatic way to carry out some of its cognitive processes (a hypothesis for which there is no current evidence), this doesn't imply that these quantum phenomena are *necessary* in order to carry out the given cognitive processes. For example there is evidence that birds use quantum nonlocal phenomena to carry out navigation based on the Earth's magnetic fields [**?**]; yet scientists have built instruments that carry out the same functions without using any special quantum effects.

Quantum "magic" aside, it is also conceivable that building AGI is fundamentally impossible for some *other* reason we don't understand. Without getting religious about it, it is rationally quite possible that some aspects of the universe are beyond the scope of scientific methods. Science is fundamentally about recognizing patterns in finite sets of bits (e.g. finite sets of finite-precision observations), whereas mathematics recognizes many sets much larger than this. Selmer Bringsjord [**?**], and other advocates of "hypercomputing" approaches to intelligence, argue that the human mind depends on massively large infinite sets and therefore can never be simulated on digital computers nor understood via finite sets of finite-precision measurements such as science deals with.

But again, while this sort of possibility is interesting to speculate about, there's no real reason to believe it at this time. Brain science and AI are both very young sciences and the "working hypothesis" that digital computers can manifest advanced AGI has hardly been explored at all yet, relative to what will be possible in the next decades as computers get more and more powerful and our understanding of neuroscience and cognitive science gets more and more complete. The CogPrime AGI design presented here is based on this working hypothesis.

Many of the ideas in the book are actually independent of the "mind can be implemented digitally" working hypothesis, and could apply to AGI systems built on analog, quantum or other non-digital frameworks – but we will not pursue these possibilities here. For the moment, outlining an AGI design for digital computers is hard enough! Regardless of speculations about quantum computing in the brain, it seems clear that AGI on quantum computers is part of our future and will be a powerful thing; but the description of a CogPrime analogue for quantum computers will be left for a later work.

1.5 Five Key Words

As noted, the CogPrime approach lies squarely in the integrative cognitive architecture camp. But it is not a haphazard or opportunistic combination of algorithms and data structures. At bottom it is motivated by the *patternist* philosophy of mind laid out in Ben Goertzel's book *The Hidden Pattern* [Goe06a], which was in large part a summary and reformulation of ideas presented in a series of books published earlier by the same author [Goe94], [?], [?], [?], [Goe01]. A few of the core ideas of this philosophy are laid out in Chapter 3, though that chapter is by no means a thorough summary.

One way to summarize some of the most important yet commonsensical parts of the patternist philosophy of mind, in an AGI context, is to list five words: **perception, memory, prediction, action, goals**.

In a phrase: **"A mind uses perception and memory to make predictions about which actions will help it achieve its goals."**

This ties in with the ideas of many other thinkers, including Jeff Hawkins' "memory/prediction" theory [HB06], and it also speaks directly to the formal characterization of intelligence presented in Chapter 7: general intelligence as "the ability to achieve complex goals in complex environments."

Naturally the goals involved in the above phrase may be explicit or implicit to the intelligent agent, and they may shift over time as the agent develops.

Perception is taken to mean pattern recognition: the recognition of (novel or familiar) patterns in the environment or in the system itself. Memory is the storage of already-recognized patterns, enabling recollection or regeneration of these patterns as needed. Action is the formation of patterns in the body and world. Prediction is the utilization of temporal patterns to guess what perceptions will be seen in the future, and what actions will achieve what effects in the future – in essence, prediction consists of temporal pattern recognition, plus the (implicit or explicit) assumption that the universe possesses a "habitual tendency" according to which previously observed patterns continue to apply.

1.5.1 Memory and Cognition in CogPrime

Each of these five concepts has a lot of depth to it, and we won't say too much about them in this brief introductory overview; but we will take a little time to say something about memory in particular.

As we'll see in Chapter 7, one of the things that the mathematical theory of general intelligence makes clear is that, if you assume your AI system has a huge amount of computational resources, then creating general intelligence is not a big trick. Given enough computing power, a very brief and simple program can achieve any computable goal in any computable environment, quite effectively. Marcus Hutter's $AIXI^{tl}$ design [Hut05] gives one way of

doing this, backed up by rigorous mathematics. Put informally, what this means is: the problem of AGI is really a problem of coping with inadequate compute resources, just as the problem of natural intelligence is really a problem of coping with inadequate energetic resources.

One of the key ideas underlying CogPrime is a principle called *cognitive synergy*, which explains how real-world minds achieve general intelligence using limited resources, by appropriately organizing and utilizing their memories.

This principle says that there are many different kinds of memory in the mind: sensory, episodic, procedural, declarative, attentional, intentional. Each of them has certain learning processes associated with it; for example, reasoning is associated with declarative memory. And the synergy part is that the learning processes associated with each kind of memory have got to help each other out when they get stuck, rather than working at cross-purposes.

Cognitive synergy is a fundamental principle of *general intelligence* – it doesn't tend to play a central role when you're building narrow-AI systems.

In the CogPrime approach all the different kinds of memory are linked together in a single meta-representation, a sort of combined semantic/neural network called the AtomSpace. It represents everything from perceptions and actions to abstract relationships and concepts and even a system's model of itself and others. When specialized representations are used for other types of knowledge (e.g. program trees for procedural knowledge, spatiotemporal hierarchies for perceptual knowledge) then the knowledge stored outside the AtomSpace is represented via tokens (Atoms) in the AtomSpace, allowing it to be located by various cognitive processes, and associated with other memory items of any type.

So for instance an OpenCog AI system has an AtomSpace, plus some specialized knowledge stores linked into the AtomSpace; and it also has specific algorithms acting on the AtomSpace and appropriate specialized stores corresponding to each type of memory. Each of these algorithms is complex and has its own story; for instance (an incomplete list, for more detail see the following section of this Introduction):

- Declarative knowledge is handled using Probabilistic Logic Networks, described in Chapter 34 and others;
- Procedural knowledge is handled using MOSES, a probabilistic evolutionary learning algorithm described in Chapter 21 and others;
- Attentional knowledge is handled by ECAN (economic attention allocation), described in Chapter 23 and others;
- OpenCog contains a language comprehension system called RelEx that takes English sentences and turns them into nodes and links in the AtomSpace. It's currently being extended to handle Chinese. RelEx handles mostly declarative knowledge but also involves some procedural knowledge for linguistic phenomena like reference resolution and semantic disambiguation.

But the crux of the CogPrime cognitive architecture is not any particular cognitive process, but rather the way they all work together using cognitive synergy.

1.6 Virtually and Robotically Embodied AI

Another issue that will arise frequently in these pages is embodiment. There's a lot of debate in the AI community over whether embodiment is necessary for advanced AGI or not. Personally, we doubt it's necessary but we think it's extremely convenient, and are thus considerably interested in both virtual world and robotic embodiment. The CogPrime architecture itself is neutral on the issue of embodiment, and it could be used to build a mathematical theorem prover or an intelligent chat bot just as easily as an embodied AGI system. However, most of our attention has gone into figuring out how to use CogPrime to control embodied agents in virtual worlds, or else (to a lesser extent) physical robots. For instance, during 2011-2012 we are involved in a Hong Kong government funded project using OpenCog to control video game agents in a simple game world modeled on the game Minecraft [**?**].

Current virtual world technology has significant limitations that make them far less than ideal from an AGI perspective, and in Chapter 16 we will discuss how they can be remedied. However, for the medium-term future virtual worlds are not going to match the natural world in terms of richness and complexity – and so there's also something to be said for physical robots that interact with all the messiness of the real world.

With this in mind, in the Artificial Brain Lab at Xiamen University in 2009-2010, we conducted some experiments using OpenCog to control the Nao humanoid robot [GD09]. The goal of that work was to take the same code that controls the virtual dog and use it to control the physical robot. But it's harder because in this context we need to do real vision processing and real motor control. One of the key ideas involved in this project is what's called Neuro-Symbolic architecture. For instance, one can use a hierarchical neural net for vision processing, and then link these formal neurons into the nodes and links in the AtomSpace that represent concepts. So the neural and symbolic systems can work together, a notion we will review in more detail in Chapter 26.

1.7 Language Learning

One of the subtler aspects of our current approach to teaching CogPrime is language learning. Three relatively crisp and simple approaches to language learning would be:

- Build a language processing system using hand-coded grammatical rules, based on linguistic theory;
- Train a language processing system using supervised, unsupervised or semisupervised learning, based on computational linguistics;
- Have an AI system learn language via experience, based on imitation and reinforcement and experimentation, without any built-in distinction between linguistic behaviors and other behaviors.

While the third approach is conceptually appealing, our current approach in CogPrime (described in a series of chapters in Part 2) is none of the above, but rather a combination of the above. OpenCog contains a natural language processing system built using a combination of the rule-based and statistical approaches, which has reasonably adequate functionality; and our plan is to use it as an initial condition for ongoing adaptive improvement based on embodied communicative experience.

1.8 AGI Ethics

When discussing AGI work with the general public, ethical concerns often arise. Science fiction films like the *Terminator* series have raised public awareness of the possible dangers of advanced AGI systems without correspondingly advanced ethics. Non-profit organizations like the Singularity Institute for AI ((http://singinst.org) have arisen specifically to raise attention about, and foster research on, these potential dangers.

Our main focus here is on how to create AGI, not how to teach an AGI human ethical principles. However, we will address the latter issue explicitly in Chapter 12, and we do think it's important to emphasize that AGI ethics has been at the center of the design process throughout the conception and development of CogPrime and OpenCog.

Broadly speaking there are (at least) two major threats related to advanced AGI. One is that people might use AGIs for bad ends; and the other is that, even if an AGI is made with the best intentions, it might reprogram itself in a way that causes it to do something terrible. If it's smarter than us, we might be watching it carefully while it does this, and have no idea what's going on.

The best way to deal with this second "bad AGI" problem is to build ethics into your AGI architecture – and we have done this with CogPrime , via creating a goal structure that explicitly supports ethics-directed behavior, and via creating an overall architecture that supports "ethical synergy" along with cognitive synergy. In short, the notion of ethical synergy is that there are different kinds of ethical thinking associated with the different kinds of memory and you want to be sure your AGI has all of them, and that it uses them together effectively.

In order to create AGI that is not only intelligent but beneficial to other sentient beings, ethics has got to be part of the design and the roadmap. As we teach our AGI systems, we need to lead them through a series of instructional and evaluative tasks that move from a primitive level to the mature human level – in intelligence, but also in ethical judgment.

1.9 Structure of the Book

The book is divided into two parts. The technical particulars of CogPrime are discussed in Part 2; what we deal with in Part 1 are important preliminary and related matters such as:

- The nature of real-world general intelligence, both conceptually and from the perspective of formal modeling (Section I).
- The nature of cognitive and ethical development for humans and AGIs (Section III).
- The high-level properties of CogPrime , including the overall architecture and the various sorts of memory involved (Section IV).
- What kind of path may viably lead us from here to AGI, with focus laid on preschool-type environments that easily foster humanlike cognitive development. Various advanced aspects of AGI systems, such as the network and algebraic structures that may emerge from them, the ways in which they may self-modify, and the degree to which their initial design may constrain or guide their future state even after long periods of radical self-improvement (Section V).

One point made repeatedly throughout Part 1, which is worth emphasizing here, is the current lack of a really rigorous and thorough general technical theory of general intelligence. Such a theory, if complete, would be incredibly helpful for understanding complex AGI architectures like CogPrime . Lacking such a theory, we must work on CogPrime and other such systems using a combination of theory, experiment and intuition. This is not a bad thing, but it will be very helpful if the theory and practice of AGI are able to grow collaboratively together.

1.10 Key Claims of the Book

We will wrap up this Introduction with a systematic list of some of the key claims to be argued for in these pages. Not all the terms and ideas in these claims have been mentioned in the preceding portions of this Introduction, but we hope they will be reasonably clear to the reader anyway, at least in a general sense. This list of claims will be revisited in Chapter 48 near the

end of Part 2, where we will look back at the ideas and arguments that have been put forth in favor of them, in the intervening chapters.

In essence this is a list of claims such that, if the reader accepts these claims, they should probably accept that the CogPrime approach to AGI is a viable one. On the other hand if the reader rejects one or more of these claims, they may find one or more aspects of CogPrime unacceptable for some reason.

Without further ado, now, the claims:

1. General intelligence (at the human level and ultimately beyond) can be achieved via creating a computational system that seeks to achieve its goals, via using perception and memory to predict which actions will achieve its goals in the contexts in which it finds itself.
2. To achieve general intelligence in the context of human-intelligence-friendly environments and goals using feasible computational resources, it's important that an AGI system can handle different kinds of memory (declarative, procedural, episodic, sensory, intentional, attentional) in customized but interoperable ways.
3. Cognitive synergy: It's important that the cognitive processes associated with different kinds of memory can appeal to each other for assistance in overcoming bottlenecks in a manner that enables each cognitive process to act in a manner that is sensitive to the particularities of each others' internal representations, and that doesn't impose unreasonable delays on the overall cognitive dynamics.
4. As a general principle, neither purely localized nor purely global memory is sufficient for general intelligence under feasible computational resources; "glocal" memory will be required.
5. To achieve human-like general intelligence, it's important for an intelligent agent to have sensory data and motoric affordances that roughly emulate those available to humans. We don't know exactly how close this emulation needs to be, which means that our AGI systems and platforms need to support fairly flexible experimentation with virtual-world and/or robotic infrastructures.
6. To work toward adult human-level, roughly human-like general intelligence, one fairly easily comprehensible path is to use environments and goals reminiscent of human childhood, and seek to advance one's AGI system along a path roughly comparable to that followed by human children.
7. It is most effective to teach an AGI system aimed at roughly human-like general intelligence via a mix of spontaneous learning and explicit instruction, and to instruct it via a combination of imitation, reinforcement and correction, and a combination of linguistic and nonlinguistic instruction.
8. One effective approach to teaching an AGI system human language is to supply it with some in-built linguistic facility, in the form of rule-

based and statistical-linguistics-based NLP systems, and then allow it to improve and revise this facility based on experience.

9. An AGI system with adequate mechanisms for handling the key types of knowledge mentioned above, and the capability to explicitly recognize large-scale pattern in itself, should, **upon sustained interaction with an appropriate environment in pursuit of appropriate goals**, emerge a variety of complex structures in its internal knowledge network, including, but not limited to:
   - a hierarchical network, representing both a spatiotemporal hierarchy and an approximate "default inheritance" hierarchy, cross-linked;
   - a heterarchical network of associativity, roughly aligned with the hierarchical network;
   - a self network which is an approximate micro image of the whole network;
   - inter-reflecting networks modeling self and others, reflecting a "mirrorhouse" design pattern'.
10. Given the strengths and weaknesses of current and near-future digital computers,
    a. a (loosely) neural-symbolic network is a good representation for directly storing many kinds of memory, and interfacing between those that it doesn't store directly;
    b. Uncertain logic is a good way to handle declarative knowledge. To deal with the problems facing a human-level AGI, an uncertain logic must integrate imprecise probability and fuzziness with a broad scope of logical constructs. PLN is one good realization.
    c. Programs are a good way to represent procedures (both cognitive and physical-action, but perhaps not including low-level motor-control procedures).
    d. Evolutionary program learning is a good way to handle difficult program learning problems. Probabilistic learning on normalized programs is one effective approach to evolutionary program learning. MOSES is one good realization of this approach.
    e. Multistart hill-climbing, with a strong Occam prior, is a good way to handle relatively straightforward program learning problems.
    f. Activation spreading and Hebbian learning comprise a reasonable way to handle attentional knowledge (though other approaches, with greater overhead cost, may provide better accuracy and may be appropriate in some situations).
       - Artificial economics is an effective approach to activation spreading and Hebbian learning in the context of neural-symbolic networks;
       - ECAN is one good realization of artificial economics;

- A good trade-off between comprehensiveness and efficiency is to focus on two kinds of attention: processor attention (represented in CogPrime by ShortTermImportance) and memory attention (represented in CogPrime by LongTermImportance).

g. Simulation is a good way to handle episodic knowledge (remembered and imagined). Running an internal world simulation engine is an effective way to handle simulation.
h. Hybridization of one's integrative neural-symbolic system with a spatiotemporally hierarchical deep learning system is an effective way to handle representation and learning of low-level sensorimotor knowledge. DeSTIN is one example of a deep learning system of this nature that can be effective in this context.
i. One effective way to handle goals is to represent them declaratively, and allocate attention among them economically. CogPrime 's PLN/ECAN based framework for handling intentional knowledge is one good realization.

11. It is important for an intelligent system to have some way of recognizing large-scale patterns in itself, and then embodying these patterns as new, localized knowledge items in its memory. Given the use of a neural-symbolic network for knowledge representation, a graph-mining based "map formation" heuristic is one good way to do this.
12. Occam's Razor: Intelligence is closely tied to the creation of procedures that achieve goals in environments *in the simplest possible way*. Each of an AGI system's cognitive algorithms should embody a simplicity bias in some explicit or implicit form.
13. An AGI system, if supplied with a commonsensically ethical goal system and an intentional component based on rigorous uncertain inference, should be able to reliably achieve a much higher level of commonsensically ethical behavior than any human being.
14. Once sufficiently advanced, an AGI system with a logic-based declarative knowledge approach and a program-learning-based procedural knowledge approach should be able to radically self-improve via a variety of methods, including supercompilation and automated theorem-proving.

Section I
Artificial and Natural General Intelligence

Chapter 2
What Is Human-Like General Intelligence?

2.1 Introduction

CogPrime , the AGI architecture on which the bulk of this book focuses, is aimed at the creation of artificial general intelligence that is vaguely human-like in nature, and possesses capabilities at the human level and ultimately beyond.

Obviously this description begs some foundational questions, such as, for starters: What is "general intelligence"? What is "human-like general intelligence"? What is "intelligence" at all?

Perhaps in future there will exist a rigorous theory of general intelligence which applies usefully to real-world biological and digital intelligences. In later chapters we will give some ideas in this direction. But such a theory is currently nascent at best. So, given the present state of science, these two questions about intelligence must be handled via a combination of formal and informal methods. This brief, informal chapter attempts to explain our view on the nature of intelligence in sufficient detail to place the discussion of CogPrime in appropriate context, without trying to resolve all the subtleties.

Psychologists sometimes define human general intelligence using IQ tests and related instruments [**?**] – so one might wonder: why not just go with that? But these sorts of intelligence testing approaches have difficulty even extending to humans from diverse cultures [**?**]. So it's clear that to ground AGI approaches that are not based on precise modeling of human cognition, one requires a more fundamental understanding of the nature of general intelligence. On the other hand, if one conceives intelligence too broadly and mathematically, there's a risk of leaving the real human world too far behind. In this chapter (followed up in Chapters 9 and 7 with more rigor), we present a highly abstract understanding of intelligence-in-general, and then portray human-like general intelligence as a (particularly relevant) special case.

2.1.1 What Is General Intelligence?

Many attempts to characterize general intelligence have been made; Legg and Hutter [?] review over 70! Our preferred abstract characterization of intelligence is: **the capability of a system to choose actions maximizing its goal-achievement, based on its perceptions and memories, and making reasonably efficient use of its computational resources** [?]. A general intelligence is then understood as one that can do this for a variety of complex goals in a variety of complex environments.

However, apart from positing definitions, it is difficult to say anything nontrivial about general intelligence *in general*. Marcus Hutter [?] has demonstrated, using a characterization of general intelligence similar to the above, that a very simple algorithm called AIXI^{tl} can demonstrate arbitrarily high levels of general intelligence, if given sufficiently immense computational resources. This is interesting because it shows that (if we assume the universe can effectively be modeled as a computational system) general intelligence is basically a problem of computational efficiency. The particular structures and dynamics that characterize real-world general intelligences like humans arise because of the need to achieve reasonable levels of intelligence using modest space and time resources.

The "patternist" theory of mind presented in [Goe06a] and briefly summarized in Chapter 3 below presents a number of *emergent structures and dynamics* that are hypothesized to characterize pragmatic general intelligence, including such things as system-wide hierarchical and heterarchical knowledge networks, and a dynamic and self-maintaining self-model. Much of the thinking underlying CogPrime has centered on how to make multiple learning components combine to give rise to these emergent structures and dynamics.

2.1.2 What Is Human-like General Intelligence?

General principles like "complex goals in complex environments" and patternism are not sufficient to specify the nature of *human-like* general intelligence. Due to the harsh reality of computational resource restrictions, real-world general intelligences are necessarily biased to particular classes of environments. Human intelligence is biased toward the physical, social and linguistic environments in which humanity evolved, and if AI systems are to possess humanlike general intelligence they must to some extent share these biases.

But what are these biases, specifically? This is a large and complex question, which we seek to answer in a theoretically grounded way in Chapter 9. However, before turning to abstract theory, one may also approach the question in a pragmatic way, by looking at the categories of things that humans

do to manifest their particular variety of general intelligence. This is the task of the following section.

2.2 Commonly Recognized Aspects of Human-like Intelligence

It would be nice if we could give some sort of "standard model of human intelligence" in this chapter, to set the context for our approach to artificial general intelligence – but the truth is that there isn't any. What the cognitive science field has produced so far is better described as: a broad set of principles and platitudes, plus a long, loosely-organized list of ideas and results. Chapter 5 below constitutes an attempt to present an integrative architecture diagram for human-like general intelligence, synthesizing the ideas of a number of different AGI and cognitive theorists. However, though the diagram given there attempts to be inclusive, it nonetheless contains many features that are accepted by only a plurality of the research community.

The following list of key aspects of human-like intelligence has a better claim at truly being generic and representing the consensus understanding of contemporary science. It was produced by a very simple method: starting with the Wikipedia page for cognitive psychology, and then adding a few items onto it based on scrutinizing the tables of contents of some top-ranked cognitive psychology textbooks. There is some redundancy among list items, and perhaps also some minor omissions (depending on how broadly one construes some of the items), but the point is to give a broad indication of human mental functions as standardly identified in the psychology field:

- Perception
  - General perception
  - Psychophysics
  - Pattern recognition (the ability to correctly interpret ambiguous sensory information)
  - Object and event recognition
  - Time sensation (awareness and estimation of the passage of time)
- Motor Control
  - Motor planning
  - Motor execution
  - Sensorimotor integration
- Categorization
  - Category induction and acquisition
  - Categorical judgement and classification

  - Category representation and structure
  - Similarity

- Memory
  - Aging and memory
  - Autobiographical memory
  - Constructive memory
  - Emotion and memory
  - False memories
  - Memory biases
  - Long-term memory
  - Episodic memory
  - Semantic memory
  - Procedural memory
  - Short-term memory
  - Sensory memory
  - Working memory

- Knowledge representation
  - Mental imagery
  - Propositional encoding
  - Imagery versus propositions as representational mechanisms
  - Dual-coding theories
  - Mental models

- Language
  - Grammar and linguistics
  - Phonetics and phonology
  - Language acquisition

- Thinking
  - Choice
  - Concept formation
  - Judgment and decision making
  - Logic, formal and natural reasoning
  - Problem solving
  - Planning
  - Numerical cognition
  - Creativity

- Consciousness
  - Attention and Filtering (the ability to focus mental effort on specific stimuli whilst excluding other stimuli from consideration)

  - Access consciousness
  - Phenomenal consciousness

- Social Intelligence
  - Distributed Cognition
  - Empathy

If there's nothing surprising to you in the above list, I'm not surprised! If you've read a bit in the modern cognitive science literature, the list may even seem trivial. But it's worth reflecting that 50 years ago, no such list could have been produced with the same level of broad acceptance. And less than 100 years ago, the Western world's scientific understanding of the mind was dominated by Freudian thinking; and not too long after that, by behaviorist thinking, which argued that theorizing about what went on inside the mind made no sense, and science should focus entirely on analyzing external behavior. The progress of cognitive science hasn't made as many headlines as contemporaneous progress in neuroscience or computing hardware and software, but it's certainly been dramatic. One of the reasons that AGI is more achievable now than in the 1950s and 60s when the AI field began, is that now we understand the structures and processes characterizing human thinking a lot better.

In spite of all the theoretical and empirical progress in the cognitive science field, however, there is still no consensus among experts on how the various aspects of intelligence in the above "human intelligence feature list" are achieved and interrelated. In these pages, however, for the purpose of motivating CogPrime , we assume a broad integrative understanding roughly as follows:

- **Perception**: There is significant evidence that human visual perception occurs using a spatiotemporal hierarchy of pattern recognition modules, in which higher-level modules deal with broader spacetime regions, roughly as in the DeSTIN AGI architecture discussed in Chapter 4. Further, there is evidence that each module carries out temporal predictive pattern recognition as well as static pattern recognition. Audition likely utilizes a similar hierarchy. Olfaction may use something more like a Hopfield attractor neural network, as described in Chapter 13. The networks corresponding to different sense modalities have multiple cross-linkages, more at the upper levels than the lower, and also link richly into the parts of the mind dealing with other functions.
- **Motor Control**: This appears to be handled by a spatiotemporal hierarchy as well, in which each level of the hierarchy corresponds to higher-level (in space and time) movements. The hierarchy is very tightly linked in with the perceptual hierarchies, allowing sensorimotor learning and coordination.
- **Memory**: There appear to be multiple distinct but tightly cross-linked memory systems, corresponding to different sorts of knowledge such as

declarative (facts and beliefs), procedural, episodic, sensorimotor, attentional and intentional (goals).

- **Knowledge Representation**: There appear to be multiple base-level representational systems; at least one corresponding to each memory system, but perhaps more than that. Additionally there must be the capability to dynamically create new context-specific representational systems founded on the base representational system.
- **Language**: While there is surely some innate biasing in the human mind toward learning certain types of linguistic structure, it's also notable that language shares a great deal of structure with other aspects of intelligence like social roles [CB00] and the physical world [**?**]. Language appears to be learned based on biases toward learning certain types of relational role systems; and language processing seems a complex mix of generic reasoning and pattern recognition processes with specialized acoustic and syntactic processing routines.
- **Consciousness** is pragmatically well-understood using Baars' "global workspace" theory, in which a small subset of the mind's content is summoned at each time into a "working memory" aka "workspace" aka "attentional focus" where it is heavily processed and used to guide action selection.
- **Thinking** is a diverse combination of processes encompassing things like categorization, (crisp and uncertain) reasoning, concept creation, pattern recognition, and others; these processes must work well with all the different types of memory and must effectively integrate knowledge in the global workspace with knowledge in long-term memory.
- **Social Intelligence** seems closely tied with language and also with self-modeling; we model ourselves in large part using the same specialized biases we use to help us model others.

None of the points in the above bullet list is particularly controversial, but neither are any of them universally agreed-upon by experts. However, in order to make any progress on AGI design one must make some commitments to particular cognition-theoretic understandings, at this level and ultimately at more precise levels as well. Further, general philosophical analyses like the patternist philosophy to be reviewed in the following chapter only provide limited guidance here. Patternism provides a filter for theories about specific cognitive functions – it rules out assemblages of cognitive-function-specific theories that don't fit together to yield a mind that could act effectively as a pattern-recognizing, goal-achieving system with the right internal emergent structures. But it's not a precise enough filter to serve as a sole guide for cognitive theory even at the high level.

The above list of points leads naturally into the integrative architecture diagram presented in Chapter 5. But that generic architecture diagram is fairly involved, and before presenting it, we will go through some more background regarding human-like intelligence (in the rest of this chapter), philosophy of mind (in Chapter 3) and contemporary AGI architectures (in Chapter4).

2.3 Further Characterizations of Humanlike Intelligence

We now present a few complementary approaches to characterizing the key aspects of humanlike intelligence, drawn from different perspectives in the psychology and AI literature. These different approaches all overlap substantially, which is good, yet each gives a slightly different slant.

2.3.1 Competencies Characterizing Human-like Intelligence

First we give a list of key competencies characterizing human level intelligence resulting from the the AGI Roadmap Workshop held at the University of Knoxville in October 2008 [1], which was organized by Ben Goertzel and Itamar Arel. In this list, each broad competency area is listed together with a number of specific competencies sub-areas within its scope:

1. **Perception**: vision, hearing, touch, proprioception, crossmodal
2. **Actuation**: physical skills, navigation, tool use
3. **Memory**: episodic, declarative, behavioral
4. **Learning**: imitation, reinforcement, interactive verbal instruction, written media, experimentation
5. **Reasoning**: deductive, abductive, inductive, causal, physical, associational, categorization
6. **Planning**: strategic, tactical, physical, social
7. **Attention**: visual, social, behavioral
8. **Motivation**: subgoal creation, affect-based motivation, control of emotions
9. **Emotion**: expressing emotion, understanding emotion
10. **Self**: self-awareness, self-control, other-awareness
11. **Social**: empathy, appropriate social behavior, social communication, social inference, group play, theory of mind
12. **Communication**: gestural, pictorial, verbal, language acquisition, crossmodal
13. **Quantitative**: counting, grounded arithmetic, comparison, measurement
14. **Building/Creation**: concept formation, verbal invention, physical construction, social group formation

[1] See http://www.ece.utk.edu/~itamar/AGI_Roadmap.html; participants included: Sam Adams, IBM Research; Ben Goertzel, Novamente LLC; Itamar Arel, University of Tennessee; Joscha Bach, Institute of Cognitive Science, University of Osnabruck, Germany; Robert Coop, University of Tennessee; Rod Furlan, Singularity Institute; Matthias Scheutz, Indiana University; J. Storrs Hall, Foresight Institute; Alexei Samsonovich, George Mason University; Matt Schlesinger, Southern Illinois University; John Sowa, Vivomind Intelligence, Inc.; Stuart C. Shapiro, University at Buffalo

Clearly this list is getting at the same things as the textbook headings given in Section 2.2, but with a different emphasis due to its origin among AGI researchers rather than cognitive psychologists. As part of the AGI Roadmap project, specific tasks were created corresponding to each of the sub-areas in the above list; we will describe some of these tasks in Chapter 17.

2.3.2 Gardner's Theory of Multiple Intelligences

The diverse list of human-level "competencies" given above is reminiscent of Gardner's [**?**] multiple intelligences (MI) framework – a psychological approach to intelligence assessment based on the idea that different people have mental strengths in different high-level domains, so that intelligence tests should contain aspects that focus on each of these domains separately. MI does not contradict the "complex goals in complex environments" view of intelligence, but rather may be interpreted as making specific commitments regarding which complex tasks and which complex environments are most important for roughly human-like intelligence.

MI does not seek an extreme generality, in the sense that it explicitly focuses on domains in which humans have strong innate capability as well as general-intelligence capability; there could easily be non-human intelligences that would exceed humans according to both the commonsense human notion of "general intelligence" and the generic "complex goals in complex environments" or Hutter/Legg-style definitions, yet would not equal humans on the MI criteria. This strong anthropocentrism of MI is not a problem from an AGI perspective so long as one uses MI in an appropriate way, i.e. only for assessing the extent to which an AGI system displays specifically *human-like* general intelligence. This restrictiveness is the price one pays for having an easily articulable and relatively easily implementable evaluation framework.

Table 2.3.2 summarizes the types of intelligence included in Gardner's MI theory.

2.3.3 Newell's Criteria for a Human Cognitive Architecture

Finally, another related perspective is given by Alan Newell's "functional criteria for a human cognitive architecture" [**?**], which require that a humanlike AGI system should:

1. Behave as an (almost) arbitrary function of the environment
2. Operate in real time
3. Exhibit rational, i.e., effective adaptive behavior

Intelligence Type	Aspects
Linguistic	Words and language, written and spoken; retention, interpretation and explanation of ideas and information via language; understands relationship between communication and meaning
Logical-Mathematical	Logical thinking, detecting patterns, scientific reasoning and deduction; analyse problems, perform mathematical calculations, understands relationship between cause and effect towards a tangible outcome
Musical	Musical ability, awareness, appreciation and use of sound; recognition of tonal and rhythmic patterns, understands relationship between sound and feeling
Bodily-Kinesthetic	Body movement control, manual dexterity, physical agility and balance; eye and body coordination
Spatial-Visual	Visual and spatial perception; interpretation and creation of images; pictorial imagination and expression; understands relationship between images and meanings, and between space and effect
Interpersonal	Perception of other people's feelings; relates to others; interpretation of behaviour and communications; understands relationships between people and their situations

Table 2.1 Types of Intelligence in Gardner's Multiple Intelligence Theory

4. Use vast amounts of knowledge about the environment
5. Behave robustly in the face of error, the unexpected, and the unknown
6. Integrate diverse knowledge
7. Use (natural) language
8. Exhibit self-awareness and a sense of self
9. Learn from its environment
10. Acquire capabilities through development
11. Arise through evolution
12. Be realizable within the brain

In our view, Newell's criterion 1 is poorly-formulated, for while universal Turing computing power is easy to come by, any finite AI system must inevitably be heavily adapted to some particular class of environments for straightforward mathematical reasons [**?**, **?**]. On the other hand, his criteria 11 and 12 are not relevant to the CogPrime approach as we are not doing biological modeling but rather AGI engineering. However, Newell's criteria 2-10 are essential in our view, and all will be covered in the following chapters.

2.4 Preschool as a View into Human-like General Intelligence

One issue that arises when pursuing the grand goal of human-level general intelligence is how to measure partial progress. The classic Turing Test of imitating human conversation remains too difficult to usefully motivate immediate-term AI research (see [?, ?] for arguments that it has been counterproductive for the AI field). The same holds true for comparable alternatives like the Robot College Test of creating a robot that can attend a semester of university and obtain passing grades. However, some researchers have suggested intermediary goals, that constitute partial progress toward the grand goal and yet are qualitatively different from the highly specialized problems to which most current AI systems are applied.

In this vein, Sam Adams and his team at IBM have outlined a so-called "Toddler Turing Test," in which one seeks to use AI to control a robot qualitatively displaying similar cognitive behaviors to a young human child (say, a 3 year old) [?]. In fact this sort of idea has a long and venerable history in the AI field – Alan Turing's original 1950 paper on AI [?], where he proposed the Turing Test, contains the suggestion that

> *"Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's?"*

We find this childlike cognition based approach promising for many reasons, including its integrative nature: what a young child does involves a combination of perception, actuation, linguistic and pictorial communication, social interaction, conceptual problem solving and creative imagination. Specifically, inspired by these ideas, in Chapter 16 we will suggest the approach of teaching and testing early-stage AGI systems in environments that emulate the preschools used for teaching human children.

Human intelligence evolved in response to the demands of richly interactive environments, and a preschool is specifically designed to be a richly interactive environment with the capability to stimulate diverse mental growth. So, we are currently exploring the use of CogPrime to control virtual agents in preschool-like virtual world environments, as well as commercial humanoid robot platforms such as the Nao (see Figure 2.1) or Robokind (2.2) in physical preschool-like robot labs.

Another advantage of focusing on childlike cognition is that child psychologists have created a variety of instruments for measuring child intelligence. In Chapter 17, we will discuss an approach to evaluating the general intelligence of human childlike AGI systems via combining tests typically used to measure the intelligence of young human children, with additional tests crafted based on cognitive science and the standard preschool curriculum.

2.4.1 Design for an AGI Preschool

More precisely, we don't suggest to place a CogPrime system in an environment that is an exact imitation of a human preschool – this would be inappropriate since current robotic or virtual bodies are very differently abled than the body of a young human child. But we aim to place CogPrime in an environment emulating the basic diversity and educational character of a typical human preschool. We stress this now, at this early point in the book, because we will use running examples throughout the book drawn from the preschool context.

The key notion in modern preschool design is the "learning center," an area designed and outfitted with appropriate materials for teaching a specific skill. Learning centers are designed to encourage learning by doing, which greatly facilitates learning processes based on reinforcement, imitation and correction; and also to provide multiple techniques for teaching the same skills, to accommodate different learning styles and prevent overfitting and overspecialization in the learning of new skills.

Centers are also designed to cross-develop related skills. A "manipulatives center," for example, provides physical objects such as drawing implements, toys and puzzles, to facilitate development of motor manipulation, visual discrimination, and (through sequencing and classification games) basic logical reasoning. A "dramatics center" cross-trains interpersonal and empathetic skills along with bodily-kinesthetic, linguistic, and musical skills. Other centers, such as art, reading, writing, science and math centers are also designed to train not just one area, but to center around a primary intelligence type while also cross-developing related areas. For specific examples of the learning centers associated with particular contemporary preschools, see [**?**]. In many progressive, student-centered preschools, students are left largely to their own devices to move from one center to another throughout the preschool room. Generally, each center will be staffed by an instructor at some points in the day but not others, providing a variety of learning experiences.

To imitate the general character of a human preschool, we will create several centers in our robot lab. The precise architecture will be adapted via experience but initial centers will likely be:

- **a blocks center**: a table with blocks on it
- **a language center**: a circle of chairs, intended for people to sit around and talk with the robot
- **a manipulatives center**, with a variety of different objects of different shapes and sizes, intended to teach visual and motor skills
- **a ball play center**: where balls are kept in chests and there is space for the robot to kick the balls around
- **a dramatics center** where the robot can observe and enact various movements

One Running Example

As we proceed through the various component structures and dynamics of CogPrime in the following chapters, it will be useful to have a few running examples to use to explain how the various parts of the system are supposed to work. One example we will use fairly frequently is drawn from the preschool context: the somewhat open-ended task of **Build me something out of blocks, that you haven't built for me before, and then tell me what it is**. This is a relatively simple task that combines multiple aspects of cognition in a richly interconnected way, and is the sort of thing that young children will naturally do in a preschool setting.

2.5 Integrative and Synergetic Approaches to Artificial General Intelligence

In Chapter 1 we characterized CogPrime as an integrative approach. And we suggest that the naturalness of integrative approaches to AGI follows directly from comparing above lists of capabilities and criteria to the array of available AI technologies. No single known algorithm or data structure appears easily capable of carrying out all these functions, so if one wants to proceed *now* with creating a general intelligence that is even vaguely humanlike, one must integrate various AI technologies within some sort of unifying architecture.

For this reason and others, an increasing amount of work in the AI community these days is integrative in one sense or another. Estimation of Distribution Algorithms integrate probabilistic reasoning with evolutionary learning [**?**]. Markov Logic Networks [**?**] integrate formal logic and probabilistic inference, as does the Probabilistic Logic Networks framework [GIGH08] utilized in CogPrime and explained further in the book, and other works in the "Progic" area such as [**?**]. Leslie Pack Kaelbling has synthesized low-level robotics methods (particle filtering) with logical inference [**?**]. Dozens of further examples could be given. The construction of practical robotic systems like the Stanley system that won the DARPA Grand Challenge [**?**] involve the integration of numerous components based on different principles. These algorithmic and pragmatic innovations provide ample raw materials for the construction of integrative cognitive architectures and are part of the reason why childlike AGI is more approachable now than it was 50 or even 10 years ago.

Further, many of the *cognitive architectures* described in the current AI literature are "integrative" in the sense of combining multiple, qualitatively different, interoperating algorithms. Chapter 4 gives a high-level overview of existing cognitive architectures, dividing them into *symbolic*, *emergentist* (e.g. neural network) and *hybrid* architectures. The hybrid architectures generally integrate symbolic and neural components, often with multiple sub-

components within each of these broad categories. However, we believe that even these excellent architectures are not integrative enough, in the sense that they lack sufficiently rich and nuanced interactions between the learning components associated with different kinds of memory, and hence are unlikely to give rise to the emergent structures and dynamics characterizing general intelligence. One of the central ideas underlying CogPrime is that with an integrative cognitive architecture that combines multiple aspects of intelligence, achieved by diverse structures and algorithms, within a common framework designed specifically to support robust **synergetic interactions** between these aspects.

The simplest way to create an integrative AI architecture is to loosely couple multiple components carrying out various functions, in such a way that the different components pass inputs and outputs amongst each other but do not interfere with or modulate each others' internal functioning in real-time. However, the human brain appears to be integrative in a much tighter sense, involving rich real-time dynamical coupling between various components with distinct but related functions. In [**?**] we have hypothesized that the brain displays a property of **cognitive synergy**, according to which multiple learning processes can not only **dispatch subproblems** to each other, but also **share contextual understanding in real-time**, so that each one can get help from the others in a contextually savvy way. By imbuing AI architectures with cognitive synergy, we hypothesize, one can get past the bottlenecks that have plagued AI in the past. Part of the reasoning here, as elaborated in Chapter 9 and [**?**], is that real physical and social environments display a rich dynamic interconnection between their various aspects, so that richly dynamically interconnected integrative AI architectures will be able to achieve goals within them more effectively.

And this brings us to the patternist perspective on intelligent systems, alluded to above and fleshed out further in Chapter 3 with its focus on the emergence of hierarchically and heterarchically structured networks of patterns, and pattern-systems modeling self and others. Ultimately the purpose of cognitive synergy in an AGI system is to enable the various AI algorithms and structures composing the system to work together effectively enough to give rise to the right *system-wide emergent structures* characterizing real-world general intelligence. The underlying theory is that intelligence is not reliant on any particular structure or algorithm, but *is* reliant on the emergence of appropriately structured networks of patterns, which can then be used to guide ongoing dynamics of pattern recognition and creation. And the underlying hypothesis is that the emergence of these structures cannot be achieved by a loosely interconnected assemblage of components, no matter how sensible the architecture; it requires a tightly connected, synergetic system.

It is possible to make these theoretical ideas about cognition mathematically rigorous; for instance, Appendix B briefly presents a formal definition of cognitive synergy that has been analyzed as part of an effort to prove

theorems about the importance of cognitive synergy for giving rise to emergent system properties associated with general intelligence. However, while we have found such formal analyses valuable for clarifying our designs and understanding their qualitative properties, we have concluded that, for the present, the best way to explore our hypotheses about cognitive synergy and human-like general intelligence is empirically – via building and testing systems like CogPrime .

2.5.1 Achieving Humanlike Intelligence via Cognitive Synergy

Summing up: at the broadest level, there are four primary challenges in constructing an integrative, cognitive synergy based approach to AGI:

1. choosing an **overall cognitive architecture** that possesses adequate richness and flexibility for the task of achieving childlike cognition.
2. Choosing **appropriate AI algorithms and data structures** to fulfill each of the functions identified in the cognitive architecture (e.g. visual perception, audition, episodic memory, language generation, analogy,...)
3. Ensuring that these algorithms and structures, within the chosen cognitive architecture, are able to cooperate in such a way as to provide appropriate **coordinated, synergetic intelligent behavior** (a critical aspect since childlike cognition is an integrated functional response to the world, rather than a loosely coupled collection of capabilities.)
4. Embedding one's system in an environment that provides **sufficiently rich stimuli and interactions** to enable the system to use this cooperation to ongoingly create an intelligent internal world-model and self-model.

We argue that CogPrime
provides a viable way to address these challenges.

Fig. 2.1 The Nao humanoid robot

Fig. 2.2 The Nao humanoid robot

Chapter 3
A Patternist Philosophy of Mind

3.1 Introduction

In the last chapter we discussed human intelligence from a fairly down-to-earth perspective, looking at the particular intelligent functions that human beings carry out in their everyday lives. And we strongly feel this practical perspective is important: Without this concreteness, it's too easy for AGI research to get distracted by appealing (or frightening) abstractions of various sorts. However, it's **also** important to look at the nature of mind and intelligence from a more general and conceptual perspective, to avoid falling into an approach that follows the particulars of human capability but ignores the deeper structures and dynamics of mind that ultimately allow human minds to be so capable. In this chapter we very briefly review some ideas from the **patternist philosophy of mind**, a general conceptual framework on intelligence which has been inspirational for many key aspects of the CogPrime design, and which has been ongoingly developed by one of the authors (Ben Goertzel) during the last two decades (in a series of publications beginning in 1991, most recently *The Hidden Pattern* [Goe06a]). Some of the ideas described are quite broad and conceptual, and are related to CogPrime only via serving as general inspirations; others are more concrete and technical, and are actually utilized within the design itself.

CogPrime is an integrative design formed via the combination of a number of different philosophical, scientific and engineering ideas. The success or failure of the design doesn't depend on any particular philosophical understanding of intelligence. In that sense, the more abstract notions presented in this chapter should be considered "optional" rather than critical in a CogPrime context. However, due to the core role patternism has played in the development of CogPrime , understanding a few things about general patternist philosophy will be helpful for understanding CogPrime , even for those readers who are not philosophically inclined. Those readers who *are* philo-

sophically inclined, on the other hand, are urged to read *The Hidden Pattern* and then interpret the particulars of CogPrime in this light.

3.2 Some Patternist Principles

The patternist philosophy of mind is a general approach to thinking about intelligent systems. It is based on the very simple premise that mind is made of pattern – and that a mind is a system for recognizing patterns in itself and the world, critically including patterns regarding which procedures are likely to lead to the achievement of which goals in which contexts.

Pattern as the basis of mind is not in itself is a very novel idea; this concept is present, for instance, in the 19'th-century philosophy of Charles Peirce [**?**], in the writings of contemporary philosophers Daniel Dennett [Den91] and Douglas Hofstadter [Hof79, Hof96], in Benjamin Whorf's [Who64] linguistic philosophy and Gregory Bateson's [Bat79] systems theory of mind and nature. Bateson spoke of the Metapattern: "that it is pattern which connects." In Goertzel's writings on philosophy of mind, an effort has been made to pursue this theme more thoroughly than has been done before, and to articulate in detail how various aspects of human mind and mind in general can be well-understood by explicitly adopting a patternist perspective. [1]

In the patternist perspective, "pattern" is generally defined as "representation as something simpler." Thus, for example, if one measures simplicity in terms of bit-count, then a program compressing an image would be a pattern in that image. But if one uses a simplicity measure incorporating run-time as well as bit-count, then the compressed version may or may not be a pattern in the image, depending on how one's simplicity measure weights the two factors. This definition encompasses simple repeated patterns, but also much more complex ones. While pattern theory has typically been elaborated in the context of computational theory, it is not intrinsically tied to computation; rather, it can be developed in any context where there is a notion of "representation" or "production" and a way of measuring simplicity. One just needs to be able to assess the extent to which f represents or produces X, and then to compare the simplicity of f and X; and then one can assess whether f is a pattern in X. A formalization of this notion of pattern is given in [Goe06a] and briefly summarized at the end of this chapter.

Next, in patternism the mind of an intelligent system is conceived as the (fuzzy) set of patterns in that system, and the set of patterns emergent between that system and other systems with which it interacts. The latter clause means that the patternist perspective is inclusive of notions of distributed in-

[1] In some prior writings the term "psynet model of mind" has been used to refer to the application of patternist philosophy to cognitive theory, but this term has been "deprecated" in recent publications as it seemed to introduce more confusion than clarification.

telligence [Hut96]. Basically, the mind of a system is the fuzzy set of different simplifying representations of that system that may be adopted.

Intelligence is conceived, similarly to in Marcus Hutter's [Hut05] recent work (and as elaborated informally in Chapter 2 above, and formally in Chapter 7 below), as the ability to achieve complex goals in complex environments; where complexity itself may be defined as the possession of a rich variety of patterns. A mind is thus a collection of patterns that is associated with a persistent dynamical process that achieves highly-patterned goals in highly-patterned environments.

An additional hypothesis made within the patternist philosophy of mind is that reflection is critical to intelligence. This lets us conceive an intelligent system as a dynamical system that recognizes patterns in its environment and itself, as part of its quest to achieve complex goals.

While this approach is quite general, it is not vacuous; it gives a particular structure to the tasks of analyzing and synthesizing intelligent systems. About any would-be intelligent system, we are led to ask questions such as:

- How are patterns represented in the system? That is, how does the underlying infrastructure of the system give rise to the displaying of a particular pattern in the system's behavior?
- What kinds of patterns are most compactly represented within the system?
- What kinds of patterns are most simply learned?
- What learning processes are utilized for recognizing patterns?
- What mechanisms are used to give the system the ability to introspect (so that it can recognize patterns in itself)?

Now, these same sorts of questions could be asked if one substituted the word "pattern" with other words like "knowledge" or "information". However, we have found that asking these questions in the context of pattern leads to more productive answers, avoiding unproductive byways and also tying in very nicely with the details of various existing formalisms and algorithms for knowledge representation and learning.

Among the many kinds of patterns in intelligent systems, *semiotic* patterns are particularly interesting ones. Peirce decomposed these into three categories:

- **iconic** patterns, which are patterns of contextually important internal similarity between two entities (e.g. an iconic pattern binds a picture of a person to that person)
- **indexical** patterns, which are patterns of spatiotemporal co-occurrence (e.g. an indexical pattern binds a wedding dress and a wedding)
- **symbolic** patterns, which are patterns indicating that two entities are often involved in the same relationships (e.g. a symbolic pattern between the number "5" (the symbol) and various sets of 5 objects (the entities that the symbol is taken to represent))

Of course, some patterns may span more than one of these semiotic categories; and there are also some patterns that don't fall neatly into any of these categories. But the semiotic patterns are particularly important ones; and symbolic patterns have played an especially large role in the history of AI, because of the radically different approaches different researchers have taken to handling them in their AI systems. Mathematical logic and related formalisms provide sophisticated mechanisms for combining and relating symbolic patterns ("symbols"), and some AI approaches have focused heavily on these, sometimes more so than on the identification of symbolic patterns in experience or the use of them to achieve practical goals. We will look fairly carefully at these differences in Chapter 4.

Pursuing the patternist philosophy in detail leads to a variety of particular hypotheses and conclusions about the nature of mind. Following from the view of intelligence in terms of achieving complex goals in complex environments, comes a view in which the dynamics of a cognitive system are understood to be governed by two main forces:

- self-organization, via which system dynamics cause existing system patterns to give rise to new ones
- goal-oriented behavior, which will be defined more rigorously in Chapter 7, but basically amounts to a system interacting with its environment in a way that appears like an attempt to maximize some reasonably simple function

Self-organized and goal-oriented behavior must be understood as cooperative aspects. If an agent is asked to build a surprising structure out of blocks and does so, this is goal-oriented. But the agent's ability to carry out this goal-oriented task will be greater if it has previously played around with blocks a lot in an unstructured, spontaneous way. And the "nudge toward creativity" given to it by asking it to build a surprising blocks structure may cause it to explore some novel patterns, which then feed into its future unstructured blocks play.

Based on these concepts, as argued in detail in [Goe06a], several primary dynamical principles may be posited, including:

- **Evolution** , conceived as a general process via which patterns within a large population thereof are differentially selected and used as the basis for formation of new patterns, based on some "fitness function" that is generally tied to the goals of the agent
  - *Example:* If trying to build a blocks structure that will surprise Bob, an agent may simulate several procedures for building blocks structures in its "mind's eye", assessing for each one the expected degree to which it might surprise Bob. The search through procedure space could be conducted as a form of evolution, via an algorithm such as MOSES.

- **Autopoiesis**: the process by which a system of interrelated patterns maintains its integrity, via a dynamic in which whenever one of the patterns in the system begins to decrease in intensity, some of the other patterns increase their intensity in a manner that causes the troubled pattern to increase in intensity again
  - *Example:* An agent's set of strategies for building the base of a tower, and its set of strategies for building the middle part of a tower, are likely to relate autopoietically. If the system partially forgets how to build the base of a tower, then it may regenerate this missing knowledge via using its knowledge about how to build the middle part (i.e., it knows it needs to build the base in a way that will support good middle parts). Similarly if it partially forgets how to build the middle part, then it may regenerate this missing knowledge via using its knowledge about how to build the base (i.e. it knows a good middle part should fit in well with the sorts of base it knows are good).
  - This same sort of interdependence occurs between pattern-sets containing more than two elements
  - Sometimes (as in the above example) autopoietic interdependence in the mind is tied to interdependencies in the physical world, sometimes not.
- **Association**. Patterns, when given attention, spread some of this attention to other patterns that they have previously been associated with in some way. Furthermore, there is Peirce's law of mind [**?**], which could be paraphrased in modern terms as stating that the mind is an associative memory network, whose dynamics dictate that every idea in the memory is an active agent, continually acting on those ideas with which the memory associates it.
  - *Example:* Building a blocks structure that resembles a tower, spreads attention to memories prior towers the agents has seen, and also to memories of people the agent knows has seen towers, and structures it has built at the same time as towers, structures that resemble towers in various respects, etc.
- **Differential attention allocation / credit assignment**. Patterns that have been valuable for goal-achievement are given more attention, and are encouraged to participate in giving rise to new patterns.
  - *Example:* Perhaps in a prior instance of the task "build me a surprising structure out of blocks," searching through memory for non-blocks structures that the agent has played with has proved a useful cognitive strategy. In that case, when the task is posed to the agent again, it should tend to allocate disproportionate resources to this strategy.

- **Pattern creation**. Patterns that have been valuable for goal-achievement are mutated and combined with each other to yield new patterns.
  - *Example:* Building towers has been useful in a certain context, but so has building structures with a large number of triangles. Why not build a tower out of triangles? Or maybe a vaguely tower-like structure that uses more triangles than a tower easily could?
  - *Example:* Building an elongated block structure resembling a table was successful in the past, as was building a structure resembling a very flat version of a chair. Generalizing, maybe building distorted versions of furniture is good. Or maybe it is building distorted version of *any* previously perceived objects that is good. Or maybe both, to different degrees....

Next, for a variety of reasons outlined in [Goe06a] it becomes appealing to hypothesize that the network of patterns in an intelligent system must give rise to the following large-scale emergent structures

- Hierarchical network. Patterns are habitually in relations of control over other patterns that represent more specialized aspects of themselves.
  - *Example:* The pattern associated with "tall building" has some control over the pattern associated with "tower", as the former represents a more general concept ... and "tower" has some control over "Eiffel tower", etc.
- Heterarchical network. The system retains a memory of which patterns have previously been associated with each other in any way.
  - *Example:* "Tower" and "snake" are distant in the natural pattern hierarchy, but may be associatively/heterarchically linked due to having a common elongated structure. This heterarchical linkage may be used for many things, e.g. it might inspire the creative construction of a tower with a snake's head.
- Dual network. Hierarchical and heterarchical structures are combined, with the dynamics of the two structures working together harmoniously. Among many possible ways to hierarchically organize a set of patterns, the one used should be one that causes hierarchically nearby patterns to have many meaningful heterarchical connections; and of course, there should be a tendency to search for heterarchical connections among hierarchically nearby patterns.
  - *Example:* While the set of patterns hierarchically nearby "tower" and the set of patterns heterarchically nearby "tower" will be quite different, they should still have more overlap than random pattern-sets of similar sizes. So, if looking for something else heterarchically near "tower", using the hierarchical information about "tower" should be of some use, and vice versa

  - In PLN, hierarchical relationships correspond to Atoms A and B so that $InheritanceAB$ and $InheritanceBA$ have highly dissimilar strength; and heterarchical relationships correspond to Intensional-Similarity relationships. The dual network structure then arises when intensional and extensional inheritance approximately correlate with each other, so that inference about either kind of inheritance assists with figuring out about the other kind.

- Self structure. A portion of the network of patterns forms into an approximate image of the overall network of patterns.
  - *Example:* Each time the agent builds a certain structure, it observes itself building the structure, and its role as "builder of a tall tower" (or whatever the structure is) becomes part of its self-model. Then when it is asked to build something new, it may consult its self-model to see if it believes itself capable of building that sort of thing (for instance, if it is asked to build something very large, its self-model may tell it that it lacks persistence for such projects, so it may reply "I can try, but I may wind up not finishing it").

As we proceed through the CogPrime design in the following pages, we will see how each of these abstract concepts arises concretely from CogPrime 's structures and algorithms. If the theory of [Goe06a] is correct, then the success of CogPrime as a design will depend largely on whether these high-level structures and dynamics can be made to emerge from the synergetic interaction of CogPrime 's representation and algorithms, when they are utilized to control an appropriate agent in an appropriate environment.

3.3 Cognitive Synergy

Now we dig a little deeper and present a different sort of "general principle of feasible general intelligence", already hinted in earlier chapters: the *cognitive synergy* principle [2], which is both a conceptual hypothesis about the structure of generally intelligent systems in certain classes of environments, and a design principle used to guide the design of CogPrime . Chapter 8 presents a mathematical formalization of the notion of cognitive synergy; here we present the conceptual idea informally, which makes it more easily digestible but also more vague-sounding.

We will focus here on cognitive synergy specifically in the case of "multi-memory systems," which we define as intelligent systems whose combination

[2] While these points are implicit in the theory of mind given in [Goe06a], they are not articulated in this specific form there. So the material presented in this section is a new development within patternist philosophy, developed since [Goe06a] in a series of conference papers such as [Goe09a].

of environment, embodiment and motivational system make it important for them to possess memories that divide into partially but not wholly distinct components corresponding to the categories of:

- Declarative memory
  - *Examples of declarative knowledge:* Towers on average are taller than buildings. I generally am better at building structures I imagine, than at imitating structures I'm shown in pictures.
- Procedural memory (memory about how to do certain things)
  - *Examples of procedural knowledge:* Practical know-how regarding how to pick up an elongated rectangular block, or a square one. Know-how regarding when to approach a problem by asking "What would one of my teachers do in this situation" versus by thinking through the problem from first principles.
- Sensory and episodic memory
  - *Example of sensory knowledge:* memory of Bob's face; memory of what a specific tall blocks tower looked like
  - *Example of episodic knowledge:* memory of the situation in which the agent first met Bob; memory of a situation in which a specific tall blocks tower was built
- Attentional memory (knowledge about what to pay attention to in what contexts)
  - *Example of attentional knowledge:* When involved with a new person, it's useful to pay attention to whatever that person looks at
- Intentional memory (knowledge about the system's own goals and subgoals)
  - *Example of intentional knowledge:* If my goal is to please some person whom I don't know that well, then a subgoal may be figuring out what makes that person smile.

In Chapter 9 below we present a detailed argument as to how the requirement for a multi-memory underpinning for general intelligence emerges from certain underlying assumptions regarding the measurement of the simplicity of goals and environments. Specifically we argue that each of these memory types corresponds to certain *modes of communication*, so that intelligent agents which have to efficiently handle a sufficient variety of types of communication with other agents, are going to have to handle all these types of memory. These types of communication overlap and are often used together, which implies that the different memories and their associated cognitive processes need to work together. The points made in this section do not rely on that argument regarding the relation of multiple memory types to the

environmental situation of multiple communication types. What they do rely on is the assumption that, in the intelligence agent in question, the different components of memory are significantly but not wholly distinct. That is, there are significant "family resemblances" between the memories of a single type, yet there are also thoroughgoing connections between memories of different types.

Repeating the above points in a slightly more organized manner and then extending them, the essential idea of cognitive synergy, in the context of multi-memory systems, may be expressed in terms of the following points

1. Intelligence, relative to a certain set of environments, may be understood as the capability to achieve complex goals in these environments.
2. With respect to certain classes of goals and environments, an intelligent system requires a "multi-memory" architecture, meaning the possession of a number of specialized yet interconnected knowledge types, including: declarative, procedural, attentional, sensory, episodic and intentional (goal-related). These knowledge types may be viewed as different sorts of pattern that a system recognizes in itself and its environment.
3. Such a system must possess knowledge creation (i.e. pattern recognition / formation) mechanisms corresponding to each of these memory types. These mechanisms are also called "cognitive processes."
4. Each of these cognitive processes, to be effective, must have the capability to recognize when it lacks the information to perform effectively on its own; and in this case, to dynamically and interactively draw information from knowledge creation mechanisms dealing with other types of knowledge
5. This cross-mechanism interaction must have the result of enabling the knowledge creation mechanisms to perform much more effectively in combination than they would if operated non-interactively. This is "cognitive synergy."

Interactions as mentioned in Points 4 and 5 in the above list are the real conceptual meat of the cognitive synergy idea. One way to express the key idea here, in an AI context, is that most AI algorithms suffer from combinatorial explosions: the number of possible elements to be combined in a synthesis or analysis is just too great, and the algorithms are unable to filter through all the possibilities, given the lack of intrinsic constraint that comes along with a "general intelligence" context (as opposed to a narrow-AI problem like chess-playing, where the context is constrained and hence restricts the scope of possible combinations that needs to be considered). In an AGI architecture based on cognitive synergy, the different learning mechanisms must be designed specifically to interact in such a way as to palliate each others' combinatorial explosions - so that, for instance, each learning mechanism dealing with a certain sort of knowledge, must synergize with learning mechanisms dealing with the other sorts of knowledge, in a way that decreases the severity of combinatorial explosion.

One prerequisite for cognitive synergy to work is that each learning mechanism must recognize when it is "stuck," meaning it's in a situation where it has inadequate information to make a confident judgment about what steps to take next. Then, when it does recognize that it's stuck, it may request help from other, complementary cognitive mechanisms.

3.4 The General Structure of Cognitive Dynamics: Analysis and Synthesis

We have discussed the need for synergetic interrelation between cognitive processes corresponding to different types of memory ... and the general high-level cognitive dynamics that a mind must possess (evolution, autopoiesis). The next step is to dig further into the nature of the cognitive processes associated with different memory types and how they give rise to the needed high-level cognitive dynamics. In this section we present a *general theory of cognitive processes* based on a decomposition of cognitive processes into the two categories of *analysis* and *synthesis*, and a general formulation of each of these categories [3].

Specifically we focus here on what we call *focused cognitive processes*; that is, cognitive processes that selectively focus attention on a subset of the patterns making up a mind. In general these are not the only kind, there may also be *global cognitive processes* that act on every pattern in a mind. An example of a global cognitive process in CogPrime is the basic attention allocation process, which spreads "importance" among all knowledge in the system's memory. Global cognitive processes are also important, but focused cognitive processes are subtler to understand which is why we spend more time on them here.

3.4.1 Component-Systems and Self-Generating Systems

We begin with autopoesis – and, more specifically, with the concept of a "component-system", as described in George Kampis's book *Self-Modifying Systems in Biology and Cognitive Science* [Kam91], and as modified into the concept of a "self-generating system" or SGS in Goertzel's book *Chaotic Logic* [Goe94]. Roughly speaking, a Kampis-style component-system consists of a set of components that combine with each other to form other, compound components. The metaphor Kampis uses is that of Lego blocks, combining to form bigger Lego structures. Compound structures may in turn be com-

[3] While these points are highly compatible with theory of mind given in [Goe06a], they are not articulated there. The material presented in this section is a new development within patternist philosophy, presented previously only in the article [GPPG06].

bined together to form yet bigger compound structures. A self-generating system is basically the same concept as a component-system, but understood to be computable, whereas Kampis claims that component-systems are uncomputable.

Next, in SGS theory there is also a notion of reduction (not present in the Lego metaphor): sometimes when components are combined in a certain way, a "reaction" happens, which may lead to the elimination of some of the components. One relevant metaphor here is chemistry. Another is abstract algebra: for instance, if we combine a component f with its "inverse" component f^{-1}, both components are eliminated. Thus, we may think about two stages in the interaction of sets of components: combination, and reduction. Reduction may be thought of as algebraic simplification, governed by a set of rules that apply to a newly created compound component, based on the components that are assembled within it.

Formally, suppose $C_1, C_2, ...$ is the set of components present in a discrete-time component-system at time t. Then, the components present at time $t+1$ are a subset of the set of components of the form

$$Reduce(Join(C_i(1), ..., C_i(r)))$$

where *Join* is a joining operation, and *Reduce* is a reduction operator. The joining operation is assumed to map tuples of components into components, and the reduction operator is assumed to map the space of components into itself. Of course, the specific nature of a component system is totally dependent on the particular definitions of the reduction and joining operators; in following chapters we will specify these for the CogPrime system, but for the purpose of the broader theoretical discussion in this section they may be left general.

What is called the "cognitive equation" in *Chaotic Logic* [**?**] is the case of a SGS where the patterns in the system at time t have a tendency to correspond to components of the system at future times $t + s$. So, part of the action of the system is to transform implicit knowledge (patterns among system components) into explicit knowledge (specific system components). We will see one version of this phenomenon in Chapter 14 where we model implicit knowledge using mathematical structures called "derived hypergraphs"; and we will also later review several ways in which CogPrime 's dynamics explicitly encourage cognitive-equation type dynamics, e.g.:

- inference, which takes conclusions implicit in the combination of logical relationships, and makes them implicit by deriving new logical relationships from them
- map formation, which takes concepts that have often been active together, and creates new concepts grouping them
- association learning, which creates links representing patterns of association between entities

- probabilistic procedure learning, which creates new models embodying patterns regarding which procedures tend to perform well according to particular fitness functions

3.4.2 Analysis and Synthesis

Now we move on to the main point of this section: the argument that all or nearly all focused cognitive processes are expressible using two general process-schemata we call *synthesis* and *analysis* [4]. The notion of "focused cognitive process" will be exemplified more thoroughly below, but in essence what is meant is a cognitive process that begins with a small number of items (drawn from memory) as its focus, and has as its goal discovering something about these items, or discovering something about something else in the context of these items or in a way strongly biased by these items. This is different from a global cognitive process whose goal is more broadly-based and explicitly involves all or a large percentage of the knowledge in an intelligent system's memory store.

Among the focused cognitive processes are those governed by the so-called *cognitive schematic* implication

$$Context \wedge Procedure \rightarrow Goal$$

where the Context involves sensory, episodic and/or declarative knowledge; and attentional knowledge is used to regulate how much resource is given to each such schematic implication in memory. Synergy among the learning processes dealing with the context, the procedure and the goal is critical to the adequate execution of the cognitive schematic using feasible computational resources. This sort of explicitly goal-driven cognition plays a significant though not necessarily dominant role in CogPrime , and is also related to production rules systems and other traditional AI systems, as will be articulated in Chapter 4.

The synthesis and analysis processes as we conceive them, in the general framework of SGS theory, are as follows. First, synthesis, as shown in Figure 3.1, is defined as

synthesis: Iteratively build compounds from the initial component pool using the combinators, greedily seeking compounds that seem likely to achieve the goal.

Or in more detail

[4] In [GPPG06], what is here called "analysis" was called "backward synthesis", a name which has some advantages since it indicated that what's happening is a form of creation; but here we have opted for the more traditional analysis/synthesis terminology

1. Begin with some initial components (the initial "current pool"), an additional set of components identified as "combinators" (combination operators), and a goal function
2. Combine the components in the current pool, utilizing the combinators, to form product components in various ways, carrying out reductions as appropriate, and calculating relevant quantities associated with components as needed
3. Select the product components that seem most promising according to the goal function, and add these to the current pool (or else simply define these as the current pool)
4. Return to Step 2

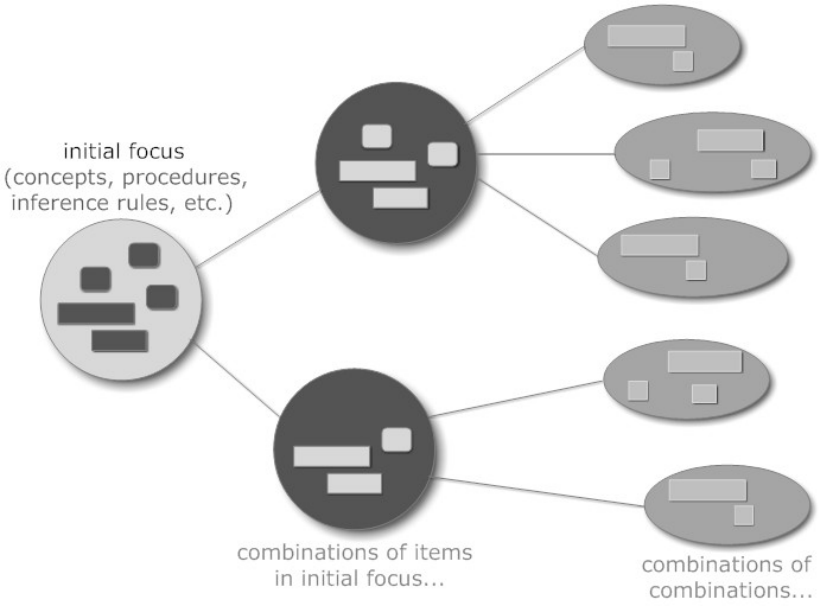


Fig. 3.1 The General Process of Synthesis

And analysis, as shown in Figure 3.2, is defined as

analysis: Iteratively search (the system's long-term memory) for component-sets that combine using the combinators to form the initial component pool (or subsets thereof), greedily seeking component-sets that seem likely to achieve the goal

or in more detail

1. Begin with some components (the initial "current pool"), and a goal function

2. Seek components so that, if one combines them to form product components using the combinators and then performs appropriate reductions, one obtains (as many as possible of) the components in the current pool
3. Use the newly found constructions of the components in the current pool, to update the quantitative properties of the components in the current pool, and also (via the current pool) the quantitative properties of the components in the initial pool
4. Out of the components found in Step 2, select the ones that seem most promising according to the goal function, and add these to the current pool (or else simply define these as the current pool)
5. Return to Step 2

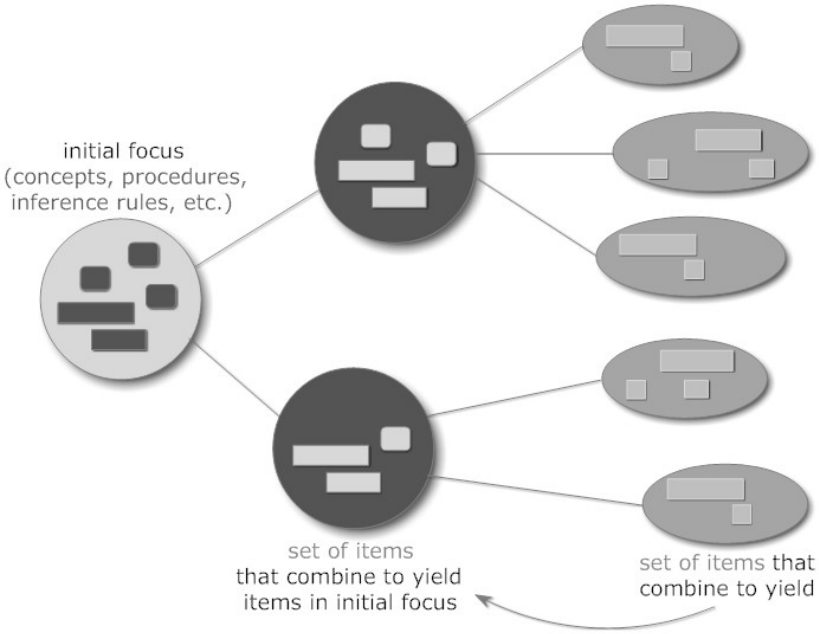


Fig. 3.2 The General Process of Analysis

More formally, synthesis may be specified as follows. Let X denote the set of combinators, and let Y_0 denote the initial pool of components (the initial focus of the cognitive process). Given Y_i, let Z_i denote the set

$$Reduce(Join(C_i(1), ..., C_i(r)))$$

where the C_i are drawn from Y_i or from X. We may then say
$Y_{i+1} = Filter(Z_i)$
where $Filter$ is a function that selects a subset of its arguments.

Analysis, on the other hand, begins with a set W of components, and a set X of combinators, and tries to find a series Y_i so that according to the process of synthesis, $Y_n{=}W$.

In practice, of course, the implementation of a synthesis process need not involve the explicit construction of the full set Z_i. Rather, the filtering operation takes place implicitly during the construction of Y_{i+1}. The result, however, is that one gets some subset of the compounds producible via joining and reduction from the set of components present in Y_i plus the combinators X.

Conceptually one may view synthesis as a very generic sort of "growth process," and analysis as a very generic sort of "figuring out how to grow something." The intuitive idea underlying the present proposal is that these forward-going and backward-going "growth processes" are among the essential foundations of cognitive control, and that a conceptually sound design for cognitive control should explicitly make use of this fact. To abstract away from the details, what these processes are about is:

- taking the general dynamic of compound-formation and reduction as outlined in Kampis and *Chaotic Logic*
- introducing goal-directed pruning ("filtering") into this dynamic so as to account for the limitations of computational resources that are a necessary part of pragmatic intelligence

3.4.3 The Dynamic of Iterative Analysis and Synthesis

While synthesis and analysis are both very useful on their own, they achieve their greatest power when harnessed together. It is my hypothesis that the dynamic pattern of alternating synthesis and analysis has a fundamental role in cognition. Put simply, synthesis creates new mental forms by combining existing ones. Then, analysis seeks simple explanations for the forms in the mind, including the newly created ones; and, this explanation itself then comprises additional new forms in the mind, to be used as fodder for the next round of synthesis. Or, to put it yet more simply:

$\Rightarrow$ **Combine** $\Rightarrow$ **Explain** $\Rightarrow$ **Combine** $\Rightarrow$ **Explain** $\Rightarrow$ **Combine** $\Rightarrow$

It is not hard to express this alternating dynamic more formally, as well.

- Let X denote any set of components.
- Let F(X) denote a set of components which is the result of synthesis on X.
- Let B(X) denote a set of components which is the result of analysis of X. We assume also a heuristic biasing the synthesis process toward simple constructs.

- Let S(t) denote a set of components at time t, representing part of a system's knowledge base.
- Let I(t) denote components resulting from the external environment at time t.

Then, we may consider a dynamical iteration of the form

$$S(t+1) = B(F(S(t) + I(t)))$$

This expresses the notion of alternating synthesis and analysis formally, as a dynamical iteration on the space of sets of components. We may then speak about attractors of this iteration: fixed points, limit cycles and strange attractors. One of the key hypotheses I wish to put forward here is that some key emergent cognitive structures are strange attractors of this equation. The iterative dynamic of combination and explanation leads to the emergence of certain complex structures that are, in essence, maintained when one recombines their parts and then seeks to explain the recombinations. These structures are built in the first place through iterative recombination and explanation, and then survive in the mind because they are conserved by this process. They then ongoingly guide the construction and destruction of various other temporary mental structures that are not so conserved.

3.4.4 Self and Focused Attention as Approximate Attractors of the Dynamic of Iterated Forward/analysis

As noted above, patternist philosophy argues that two key aspects of intelligence are emergent structures that may be called the "self" and the "attentional focus." These, it is suggested, are aspects of intelligence that may not effectively be wired into the infrastructure of an intelligent system, though of course the infrastructure may be configured in such a way as to encourage their emergence. Rather, these aspects, by their nature, are only likely to be effective if they emerge from the cooperative activity of various cognitive processes acting within a broad based of knowledge.

Above we have described the pattern of ongoing habitual oscillation between synthesis and analysis as a kind of "dynamical iteration." Here we will argue that both self and attentional focus may be viewed as strange attractors of this iteration. The mode of argument is relatively informal. The essential processes under consideration are ones that are poorly understood from an empirical perspective, due to the extreme difficulty involved in studying them experimentally. For understanding self and attentional focus, we are stuck in large part with introspection, which is famously unreliable in some contexts, yet still dramatically better than having no information at all. So, the philosophical perspective on self and attentional focus given here is a synthesis of

empirical and introspective notions, drawn largely from the published thinking and research of others but with a few original twists. From a CogPrime perspective, its use has been to guide the design process, to provide a grounding for what otherwise would have been fairly arbitrary choices.

3.4.4.1 Self

Another high-level intelligent system pattern mentioned above is the "self", which we here will tie in with analysis and synthesis processes. The term "self" as used here refers to the "phenomenal self" [Met04] or "self-model". That is, the self is the model that a system builds internally, reflecting the patterns observed in the (external and internal) world that directly pertain to the system itself. As is well known in everyday human life, self-models need not be completely accurate to be useful; and in the presence of certain psychological factors, a more accurate self-model may not necessarily be advantageous. But a self-model that is too badly inaccurate will lead to a badly-functioning system that is unable to effectively act toward the achievement of its own goals.

The value of a self-model for any intelligent system carrying out embodied agentive cognition is obvious. And beyond this, another primary use of the self is as a foundation for metaphors and analogies in various domains. Patterns recognized pertaining the self are analogically extended to other entities. In some cases this leads to conceptual pathologies, such as the anthropomorphization of trees, rocks and other such objects that one sees in some precivilized cultures. But in other cases this kind of analogy leads to robust sorts of reasoning - for instance, in reading Lakoff and Nunez's [**?**] intriguing explorations of the cognitive foundations of mathematics, it is pretty easy to see that most of the metaphors on which they hypothesize mathematics to be based, are grounded in the mind's conceptualization of itself as a spatiotemporally embedded entity, which in turn is predicated on the mind's having a conceptualization of itself (a self) in the first place.

A self-model can in many cases form a self-fulfilling prophecy (to make an obvious double-entendre'!). Actions are generated based on one's model of what sorts of actions one can and/or should take; and the results of these actions are then incorporated into one's self-model. If a self-model proves a generally bad guide to action selection, this may never be discovered, unless said self-model includes the knowledge that semi-random experimentation is often useful.

In what sense, then, may it be said that self is an attractor of iterated analysis? Analysis infers the self from observations of system behavior. The system asks: What kind of system might I be, in order to give rise to these behaviors that I observe myself carrying out? Based on asking itself this question, it constructs a model of itself, i.e. it constructs a self. Then, this self guides the system's behavior: it builds new logical relationships its self-model

and various other entities, in order to guide its future actions oriented toward achieving its goals. Based on the behaviors new induced via this constructive, forward-synthesis activity, the system may then engage in analysis again and ask: What must I be now, in order to have carried out these new actions? And so on.

Our hypothesis is that after repeated iterations of this sort, in infancy, finally during early childhood a kind of self-reinforcing attractor occurs, and we have a self-model that is resilient and doesn't change dramatically when new instances of action- or explanation-generation occur. This is not strictly a mathematical attractor, though, because over a long period of time the self may well shift significantly. But, for a mature self, many hundreds of thousands or millions of forward-analysis cycles may occur before the self-model is dramatically modified. For relatively long periods of time, small changes within the context of the existing self may suffice to allow the system to control itself intelligently.

Finally, it is interesting to speculate regarding how self may differ in future AI systems as opposed to in humans. The relative stability we see in human selves may not exist in AI systems that can self-improve and change more fundamentally and rapidly than humans can. There may be a situation in which, as soon as a system has understood itself decently, it radically modifies itself and hence violates its existing self-model. Thus: intelligence without a long-term stable self. In this case the "attractor-ish" nature of the self holds only over much shorter time scales than for human minds or human-like minds. But the alternating process of synthesis and analysis for self-construction is still critical, even though no reasonably stable self-constituting attractor ever emerges. The psychology of such intelligent systems will almost surely be beyond human beings' capacity for comprehension and empathy.

3.4.4.2 Attentional Focus

Finally, we turn to the notion of an "attentional focus" is similar to Baars' [BF09] notion of a Global Workspace, which will be reviewed in more detail in Chapter 4: a collection of mental entities that are, at a given moment, receiving far more than the usual share of an intelligent system's computational resources. Due to the amount of attention paid to items in the attentional focus, at any given moment these items are in large part driving the cognitive processes going on elsewhere in the mind as well - because the cognitive processes acting on the items in the attentional focus are often involved in other mental items, not in attentional focus, as well (and sometimes this results in pulling these other items into attentional focus). An intelligent system must constantly shift its attentional focus from one set of entities to another based on changes in its environment and based on its own shifting discoveries.

In the human mind, there is a self-reinforcing dynamic pertaining to the collection of entities in the attentional focus at any given point in time,

resulting from the observation that If A is in the attentional focus, and A and B have often been associated in the past, then odds are increased that B will soon be in the attentional focus. This basic observation has been refined tremendously via a large body of cognitive psychology work; and neurologically it follows not only from Hebb's [Heb49] classic work on neural reinforcement learning, but also from numerous more modern refinements [SB98]. But it implies that two items A and B, if both in the attentional focus, can reinforce each others' presence in the attentional focus, hence forming a kind of conspiracy to keep each other in the limelight. But of course, this kind of dynamic must be counteracted by a pragmatic tendency to remove items from the attentional focus if giving them attention is not providing sufficient utility in terms of the achievement of system goals.

The synthesis and analysis perspective provides a more systematic perspective on this self-reinforcing dynamic. synthesis occurs in the attentional focus when two or more items in the focus are combined to form new items, new relationships, new ideas. This happens continually, as one of the main purposes of the attentional focus is combinational. On the other hand, analysis then occurs when a combination that has been speculatively formed is then linked in with the remainder of the mind (the "unconscious", the vast body of knowledge that is not in the attentional focus at the given moment in time). analysis basically checks to see what support the new combination has within the existing knowledge store of the system. Thus, forward/analysis basically comes down to "generate and test", where the testing takes the form of attempting to integrate the generated structures with the ideas in the unconscious long-term memory. One of the most obvious examples of this kind of dynamic is creative thinking (Boden2003; Goertzel1997), where the attentional focus continually combinationally creates new ideas, which are then tested via checking which ones can be validated in terms of (built up from) existing knowledge.

The analysis stage may result in items being pushed out of the attentional focus, to be replaced by others. Likewise may the synthesis stage: the combinations may overshadow and then replace the things combined. However, in human minds and functional AI minds, the attentional focus will not be a complete chaos with constant turnover: sometimes the same set of ideas - or a shifting set of ideas within the same overall family of ideas – will remain in focus for a while. When this occurs it is because this set or family of ideas forms an approximate attractor for the dynamics of the attentional focus, in particular for the forward/analysis dynamic of speculative combination and integrative explanation. Often, for instance, a small "core set" of ideas will remain in the attentional focus for a while, but will not exhaust the attentional focus: the rest of the attentional focus will then, at any point in time, be occupied with other ideas related to the ones in the core set. Often this may mean that, for a while, the whole of the attentional focus will move around quasi-randomly through a "strange attractor" consisting of the set of ideas related to those in the core set.

3.4.5 Conclusion

The ideas presented above (the notions of synthesis and analysis, and the hypothesis of self and attentional focus as attractors of the iterative forward-analysis dynamic) are quite generic and are hypothetically proposed to be applicable to any cognitive system, natural or artificial. Later chapters will discuss the manifestation of the above ideas in the context of CogPrime . We have found that the analysis/synthesis approach is a valuable tool for conceptualizing CogPrime 's cognitive dynamics, and we conjecture that a similar utility may be found more generally.

Next, so as not to end the section on too blase' of a note, we will also make a stronger hypothesis: that, in order for a physical or software system to achieve intelligence that is roughly human-level in both capability and generality, using computational resources on the same order of magnitude as the human brain, this system must

- manifest the dynamic of iterated synthesis and analysis, as modes of an underlying "self-generating system" dynamic
- do so in such a way as to lead to self and attentional focus as emergent structures that serve as approximate attractors of this dynamic, over time periods are long relative to the basic "cognitive cycle time" of the system's forward/analysis dynamics

To prove the truth of a hypothesis of this nature would seem to require mathematics fairly far beyond anything that currently exists. Nonetheless, however, we feel it is important to formulate and discuss such hypotheses, so as to point the way for future investigations both theoretical and pragmatic.

3.5 Perspectives on Machine Consciousness

Finally, we can't let a chapter on philosophy – even a brief one – end without some discussion of the thorniest topic in the philosophy of mind: consciousness. Rather than seeking to resolve or comprehensively review this most delicate issue, we will restrict ourselves to giving in in Appendix [**?**] an overview of many of the common views on the subject; and here in the main text discussing the relationship between consciousness theory and and patternist philosophy of cognition, the practical work of designing and building AGI.

One fairly concrete idea about consciousness, that relates closely to certain aspects of the CogPrime design, is that the subjective experience of being conscious of some entity X, is correlated with the presence of a very intense pattern in one's overall mind-state, corresponding to X. This simple idea is also the essence of neuroscientist Susan Greenfield's theory of consciousness [Gre01] (but in her theory, "overall mind-state" is replaced with

"brain-state"), and has much deeper historical roots in philosophy of mind which we shall not venture to unravel here.

This observation relates to the idea of "moving bubbles of awareness" in intelligent systems. If an intelligent system consists of multiple processing or data elements, and during each (sufficiently long) interval of time some of these elements get much more attention than others, then one may view the system as having a certain "attentional focus" during each interval. The attentional focus is itself a significant pattern in the system (the pattern being "these elements habitually get more processor and memory", roughly speaking). As the attentional focus shifts over time one has a "moving bubble of pattern" which then corresponds experientially to a "moving bubble of awareness."

This notion of a "moving bubble of awareness" ties in very closely to global workspace theory [**?**], a cognitive theory that has broad support from neuroscience and cognitive science and has also served as the motivation for Stan Franklin's LIDA AI system [BF09], to be discussed in Chapter **??**. The global workspace theory views the mind as consisting of a large population of small, specialized processes Ð a society of agents. These agents organize themselves into coalitions, and coalitions that are relevant to contextually novel phenomena, or contextually important goals, are pulled into the global workspace (which is identified with consciousness). This workspace broadcasts the message of the coalition to all the unconscious agents, and recruits other agents into consciousness. Various sorts of contexts – e.g. goal contexts, perceptual contexts, conceptual contexts and cultural contexts – play a role in determining which coalitions are relevant, and form the unconscious ÒbackgroundÓ of the conscious global workspace. New perceptions are often, but not necessarily, pushed into the workspace. Some of the agents in the global workspace are concerned with action selection, i.e. with controlling and passing parameters to a population of possible actions. The contents of the workspace at any given time have a certain cohesiveness and interdependency, the so-called Òunity of consciousness.Ó In essence the contents of the global workspace form a moving bubble of attention or awareness.

In CogPrime , this moving bubble is achieved largely via economic attention network (ECAN) equations [GPI+10] that propagate virtual currency between nodes and links representing elements of memories, so that the attentional focus consists of the wealthiest nodes and links. Figures 3.3 and 3.4 illustrate the existence and flow of attentional focus in OpenCog. On the other hand, in Hameroff's recent model of the brain [Ham10], the the brain's moving bubble of attention is achieved through dendro-dendritic connections and the emergent dendritic web.

In this perspective, self, free will and reflective consciousness are specific phenomena occurring *within* the moving bubble of awareness. They are specific ways of experiencing awareness, corresponding to certain abstract types of physical structures and dynamics, which we shall endeavor to identify in detail in Chapter **??**.

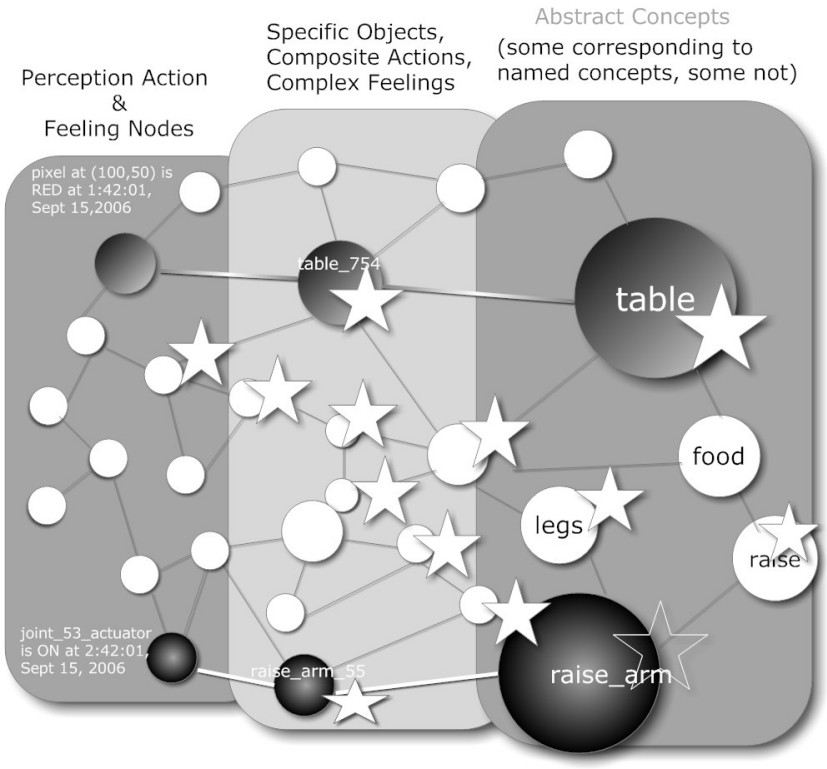


Fig. 3.3 Graphical depiction of the momentary bubble of attention in the memory of an OpenCog AI system. Circles and lines represent nodes and links in OpenCogPrimes memory, and stars denote those nodes with a high level of attention (represented in OpenCog by the ShortTermImportance node variable) at the particular point in time.

3.6 Postscript: Formalizing Pattern

Finally, before winding up our very brief tour through patternist philosophy of mind, we will briefly visit patternism's more formal side. Many of the key aspects of patternism have been rigorously formalized. Here we give only a few very basic elements of the relevant mathematics, which will be used later on in the exposition of CogPrime . (Specifically, the formal definition of pattern emerges in the CogPrime design in the definition of a fitness function for "pattern mining" algorithms and Occam-based concept creation algorithms, and the definition of intensional inheritance within PLN.)

We give some definitions, drawn from Appendix 1 of [Goe06a]:

Definition 1 *Given a metric space* (M, d)*, and two functions* $c : M \to [0, \infty]$ *(the "simplicity measure") and* $F : M \to M$ *(the "production relationship"), we say that* $\mathcal{P} \in M$ *is a* **pattern** *in* $X \in M$ *to the degree*

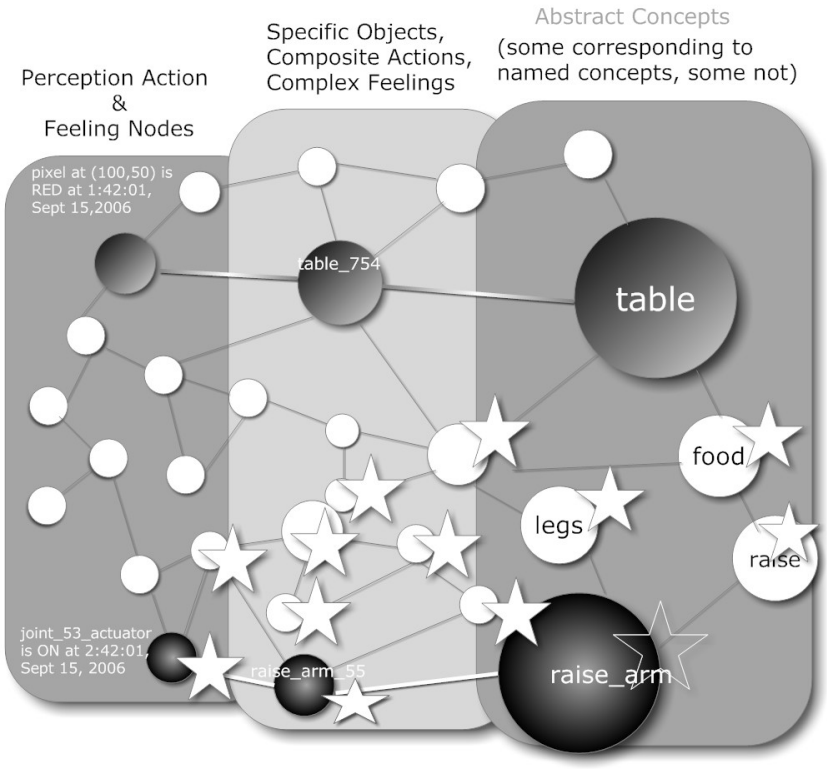


Fig. 3.4 Graphical depiction of the momentary bubble of attention in the memory of an OpenCog AI system, a few moments after the bubble shown in Figure 3.3, indicating the moving of the bubble of attention. Depictive conventions are the same as in Figure 1. This shows an idealized situation where the declarative knowledge remains invariant from one moment to the next but only the focus of attention shifts. In reality both will evolve together.

$$\iota_X^{\mathcal{P}} = \left(\left(1 - \frac{d(F(\mathcal{P}), X)}{c(X)} \right) \frac{c(X) - c(\mathcal{P})}{c(X)} \right)^+$$

This degree is called the **pattern intensity** *of* $\mathcal{P}$ *in* X*.*

For instance, if one wishes one may take c to denote algorithmic information measured on some reference Turing machine, and $F(X)$ to denote what appears on the second tape of a two-tape Turing machine t time-steps after placing X on its first tape. Other more naturalistic computational models are also possible here and are discussed extensively in Appendix 1 of [Goe06a].

Definition 2 *The* **structure** *of* $X \in M$ *is the fuzzy set* St_X *defined via the membership function*

$$\chi_{St_X}(\mathcal{P}) = \iota_X^{\mathcal{P}}$$

This lets us formalize our definition of "mind" alluded to above: the mind of X as the set of patterns associated with X. We can formalize this, for instance, by considering $\mathcal{P}$ to belong to the mind of X if it is a pattern in some Y that includes X. There are then two numbers to look at: $\iota_X^{\mathcal{P}}$ and $P(Y|X)$ (the percentage of Y that is also contained in X). To define the degree to which $\mathcal{P}$ belongs to the mind of X we can then combine these two numbers using some function f that is monotone increasing in both arguments. This highlights the somewhat arbitrary semantics of "of" in the phrase "the mind of X." Which of the patterns binding X to its environment are part of X's mind, and which are part of the world? This isn't necessarily a good question, and the answer seems to depend on what perspective you choose, represented formally in the present framework by what combination function f you choose (for instance if $f(a,b) = a^r b^{2-r}$ then it depends on the choice of $0 < r < 1$).

Next, we can formalize the notion of a "pattern space" by positing a metric on patterns, thus making pattern space a metric space, which will come in handy in some places in later chapters:

Definition 3 *Assuming M is a countable space, the* **structural distance** *is a metric d_{St} defined on M via*

$$d_{St}(X,Y) = T(\chi_{St_X}, \chi_{St_Y})$$

where T is the Tanimoto distance.

The Tanimoto distance between two real vectors A and B is defined as

$$T(A,B) = \frac{A \cdot B}{\|A\|^2 + \|B\|^2 - A \cdot B}$$

and since M is countable this can be applied to fuzzy sets such as St_X via considering the latter as vectors. (As an aside, this can be generalized to uncountable M as well, but we will not require this here.)

Using this definition of pattern, combined with the formal theory of intelligence given in Chapter [**?**], one may formalize the various hypotheses made in the previous section, regarding the emergence of different kinds of networks and structures as patterns in intelligent systems. However, it appears quite difficult to prove the formal versions of these hypotheses given current mathematical tools, which renders such formalizations of limited use.

Finally, consider the case where the metric space M has a partial ordering $<$ on it; we may then define

Definition 3.1. $\mathcal{R} \in M$ is a **subpattern** in $X \in M$ to the degree

$$\kappa_X^{\mathcal{R}} = \frac{\int_{\mathcal{P} \in M} \text{true}(R < P) d\iota_X^{\mathcal{P}}}{\int_{\mathcal{P} \in M} d\iota_X^{\mathcal{P}}}$$

This degree is called the **subpattern intensity** of $\mathcal{P}$ in X.

Roughly speaking, the subpattern intensity measures the percentage of patterns in X that contain R (where "containment" is judged by the partial ordering $<$). But the percentage is measured using a weighted average, where each pattern is weighted by its intensity as a pattern in X. A subpattern may or may not be a pattern on its own. A nonpattern that happens to occur within many patterns may be an intense subpattern.

Whether the subpatterns in X are to be considered part of the "mind" of X is a somewhat superfluous question of semantics. Here we choose to extend the definition of mind given in [Goe06a] to include subpatterns as well as patterns, because this makes it simpler to describe the relationship between hypersets and minds, as we will do in Appendix C.

Chapter 4
Brief Survey of Cognitive Architectures

4.1 Introduction

While we believe CogPrime is the most thorough attempt at an architecture for advanced AGI, to date, we certainly recognize there have been many valuable attempts in the past with similar aims; and we also have great respect for other AGI efforts occurring in parallel with CogPrime development, based on alternative, sometimes overlapping, theoretical presuppositions and practical choices. In most of this book we will ignore these other current and historical efforts except where they are directly useful for CogPrime – there are many literature reviews already published, and this is a research treatise not a textbook. In this chapter, however, we will break from this pattern and give a rough high-level overview of the various AGI architectures at play in the field today. The overview definitely has a bias toward other work with some direct relevance to CogPrime , but not an overwhelming bias; we also discuss a number of approaches that are unrelated to, and even in some cases conceptually orthogonal to, our own.

CogPrime builds on prior AI efforts in a variety of ways. Most of the specific algorithms and structures in CogPrime have their roots in prior AI work; and in addition, the CogPrime cognitive architecture has been heavily inspired by some other holistic cognitive architectures, especially (but not exclusively) MicroPsi [Bac09], LIDA [BF09] and DeSTIN [ARK09a, ARC09]. In this chapter we will briefly review some existing cognitive architectures, with especial but not exclusive emphasis on the latter three.

We will articulate some rough mappings between elements of these other architectures and elements of CogPrime – some in this chapter, and some in Chapter 5. However, these mappings will mostly be left informal and very incompletely specified. The articulation of detailed inter-architecture mappings is an important project, but would be a substantial additional project going well beyond the scope of this book. We will not give a thorough re-

view of the similarities and differences between CogPrime and each of these architectures, but only mention some of the highlights.

The reader desiring a more thorough review of cognitive architectures is referred to Wlodek Duch's review paper from the AGI-08 conference [DOP08]; and also to Alexei Samsonovich's review paper [**?**], which compares a number of cognitive architectures in terms of a feature checklist, and was created collaboratively with the creators of the architectures.

Duch, in his survey of cognitive architectures [DOP08], divides existing approaches into three paradigms – symbolic, emergentist and hybrid – as broadly indicated in 4.1. Drawing on his survey and updating slightly, we give here some key examples of each, and then explain why CogPrime represents a significantly more effective approach to embodied human-like general intelligence. In our treatment of emergentist architectures, we pay particular attention to *developmental robotics* architectures, which share considerably with CogPrime in terms of underlying philosophy, but differ via not integrating a symbolic "language and inference" component such as CogPrime includes.

In brief, we believe that the hybrid approach is the most pragmatic one given the current state of AI technology, but that the emergentist approach gets something fundamentally right, by focusing on the emergence of complex dynamics and structures from the interactions of simple components. So CogPrime is a hybrid architecture which (according to the cognitive synergy principle) binds its components together very tightly dynamically, allowing the emergence of complex dynamics and structures in the integrated system. Most other hybrid architectures are less tightly coupled and hence seem ill-suited to give rise to the needed emergent complexity. The other hybrid architectures that do possess the needed tight coupling, such as MicroPsi [**?**], strike us as underdeveloped and founded on insufficiently powerful learning algorithms.

4.2 Symbolic Cognitive Architectures

A venerable tradition in AI focuses on the physical symbol system hypothesis [**?**], which states that minds exist mainly to manipulate symbols that represent aspects of the world or themselves. A physical symbol system has the ability to input, output, store and alter symbolic entities, and to execute appropriate actions in order to reach its goals. Generally, symbolic cognitive architectures focus on "working memory" that draws on long-term memory as needed, and utilize a centralized control over perception, cognition and action. Although in principle such architectures could be arbitrarily capable (since symbolic systems have universal representational and computational power, in theory), in practice symbolic architectures tend to be weak in learning, creativity, procedure learning, and episodic and associative memory. Decades

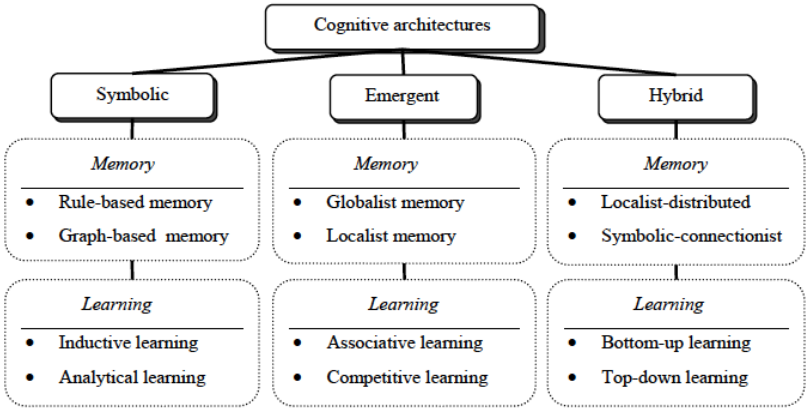


Fig. 4.1 Duch's simplified taxonomy of cognitive architectures. CogPrime falls into the "hybrid" category, but differs from other hybrid architectures in its focus on synergetic interactions between components and their potential to give rise to appropriate system-wide emergent structures enabling general intelligence.

of work in this tradition have not resolved these issues, which has led many researchers to explore other options. A few of the more important symbolic cognitive architectures are:

- **SOAR** [LRN87], a classic example of expert rule-based cognitive architecture designed to model general intelligence. It has recently been extended to handle sensorimotor functions, though in a somewhat cognitively unnatural way; and is not yet strong in areas such as episodic memory, creativity, handling uncertain knowledge, and reinforcement learning.
- **ACT-R** [AL03] is fundamentally a symbolic system, but Duch classifies it as a hybrid system because it incorporates connectionist-style activation spreading in a significant role; and there is an experimental thoroughly connectionist implementation to complement the primary mainly-symbolic implementation. Its combination of SOAR-style "production rules" with large-scale connectionist dynamics allows it to simulate a variety of human psychological phenomena, but abstract reasoning, creativity and transfer learning are still missing.
- **EPIC** [RCK01], a cognitive architecture aimed at capturing human perceptual, cognitive and motor activities through several interconnected processors working in parallel. The system is controlled by production rules for cognitive processor and a set of perceptual (visual, auditory, tactile) and motor processors operating on symbolically coded features rather than raw sensory data. It has been connected to SOAR for problem solving, planning and learning,
- **ICARUS** [Lan05], an integrated cognitive architecture for physical agents, with knowledge specified in the form of reactive skills, each de-

noting goal-relevant reactions to a class of problems. The architecture includes a number of modules: a perceptual system, a planning system, an execution system, and several memory systems. Concurrent processing is absent, attention allocation is fairly crude, and uncertain knowledge is not thoroughly handled.

- **SNePS** (Semantic Network Processing System) [SE07] is a logic, frame and network-based knowledge representation, reasoning, and acting system that has undergone over three decades of development. While it has been used for some interesting prototype experiments in language processing and virtual agent control, it has not yet been used for any large-scale or real-world application.
- **Cyc** [LG90] is an AGI architecture based on predicate logic as a knowledge representation, and using logical reasoning techniques to answer questions and derive new knowledge from old. It has been connected to a natural language engine, and designs have been created for the connection of Cyc with Albus's 4D-RCS [AM01]. Cyc's most unique aspect is the large database of commonsense knowledge that Cycorp has accumulated (millions of pieces of knowledge, entered by specially trained humans in predicate logic format); part of the philosophy underlying Cyc is that once a sufficient quantity of knowledge is accumulated in the knowledge base, the problem of creating human-level general intelligence will become much less difficult due to the ability to leverage this knowledge.

While these architectures contain many valuable ideas and have yielded some interesting results, we feel they are incapable *on their own* of giving rise to the emergent structures and dynamics required to yield humanlike general intelligence using feasible computational resources. However, we are more sanguine about the possibility of ideas and components from symbolic architectures playing a role in human-level AGI via incorporation in hybrid architectures.

We now review a few symbolic architectures in slightly more detail.

4.2.1 SOAR

The cognitive architectures best known among AI academics are probably Soar and ACT-R, both of which are explicitly being developed with the dual goals of creating human-level AGI and modeling all aspects of human psychology. Neither the Soar nor ACT-R communities feel themselves particularly near these long-term goals, yet they do take them seriously.

Soar is based on IF-THEN rules, otherwise known as "production rules." On the surface this makes it similar to old-style expert systems, but Soar is much more than an expert system; it's at minimum a sophisticated problem-solving engine. Soar explicitly conceives problem solving as a search through solution space for a "goal state" representing a (precise or approximate) problem solution. It uses a methodology of incremental search, where each step is

supposed to move the system a little closer to its problem-solving goal, and each step involves a potentially complex "decision cycle."

In the simplest case, the decision cycle has two phases:

- Gathering appropriate information from the system's long-term memory (LTM) into its working memory (WM)
- A decision procedure that uses the gathered information to decide an action

If the knowledge available in LTM isn't enough to solve the problem, then the decision procedure invokes search heuristics like hill-climbing, which try to create new knowledge (new production rules) that will help move the system closer to a solution. If a solution is found by chaining together multiple production rules, then a chunking mechanism is used to combine these rules together into a single rule for future use. One could view the chunking mechanism as a way of converting explicit knowledge into implicit knowledge, similar to "map formation" in CogPrime (see Chapter 42 of Part 2), but in the current Soar design and implementation it is a fairly crude mechanism.

In recent years Soar has acquired a number of additional methods and modalities, including some visual reasoning methods and some mechanisms for handling episodic and procedural knowledge. These expand the scope of the system but the basic production rule and chunking mechanisms as briefly described above remain the core "cognitive algorithm" of the system.

From a CogPrime perspective, what Soar offers is certainly valuable, e.g.

- heuristics for transferring knowledge from LTM into WM
- chaining and chunking of implications
- methods for interfacing between other forms of knowledge and implications

However, a very short and very partial list of the major differences between Soar and CogPrime would include

- CogPrime contains a variety of other core cognitive mechanisms beyond the management and chunking of implications
- the variety of "chunking" type methods in CogPrime goes far beyond the sort of localized chunking done in Soar
- CogPrime is committed to representing uncertainty at the base level whereas Soar's production rules are crisp
- The mechanisms for LTM-WM interaction are rather different in CogPrime , being based on complex nonlinear dynamics as represented in Economic Attention Allocation (ECAN)
- Currently Soar does not contain creativity-focused heuristics like blending or evolutionary learning in its core cognitive dynamic.

4.2.2 ACT-R

In the grand scope of cognitive architectures, ACT-R is quite similar to Soar, but there are many micro-level differences. ACT-R is defined in terms of declarative and procedural knowledge, where procedural knowledge takes the form of Soar-like production rules, and declarative knowledge takes the form of chunks. It contains a variety of mechanisms for learning new rules and chunks from old; and also contains sophisticated probabilistic equations for updating the activation levels associated with items of knowledge (these equations being roughly analogous in function to, though quite different from, the ECAN equations in CogPrime).

Figure 4.2 displays the current architecture of ACT-R. The flow of cognition in the system is in response to the current goal, currently active information from declarative memory, information attended to in perceptual modules (vision and audition are implemented), and the current state of motor modules (hand and speech are implemented). The early work with ACT-R was based on comparing system performance to human behavior, using only behavioral measures, such as the timing of keystrokes or patterns of eye movements. Using such measures, it was not possible to test detailed assumptions about which modules were active in the performance of a task. More recently the ACT-R community has been engaged in a process of using imaging data to provide converging data on module activity. Figure 4.3 illustrates the associations they have made between the modules in Figure 4.2 and brain regions. Coordination among all of these components occurs through actions of the procedural module, which is mapped to the basal ganglia.

In practice ACT-R, even more so than Soar, seems to be used more as a programming framework for cognitive modeling than as an AI system. One can fairly easily use ACT-R to program models of specific human mental behaviors, which may then be matched against psychological data. Opinions differ as to whether this sort of modeling is valuable for achieving AGI goals. CogPrime is not designed to support this kind of modeling, as it intentionally does many things very differently from humans.

ACT-R in its original form did not say much about perceptual and motor operations, but recent versions have incorporated EPIC, an independent cognitive architecture focused on modeling these aspects of human behavior.

4.2.3 Cyc and Texai

Our review of cognitive architectures would be incomplete without mentioning Cyc [LG90], one of the best known and best funded AGI-oriented projects in history. While the main focus of the Cyc project has been on the hand-coding of large amounts of declarative knowledge, there is also a cognitive architecture of sorts there. The center of Cyc is an engine for logical deduc-

Fig. 4.2 High-level architecture of ACT-R

tion, acting on knowledge represented in predicate logic. A natural language

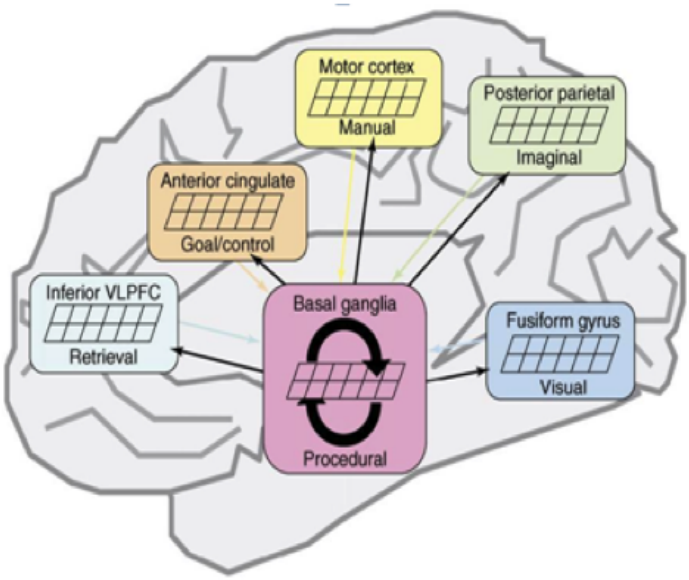


Fig. 4.3 Conjectured Mapping Between ACT-R and the Brain

engine has been associated with the logic engine, which enables one to ask English questions and get English replies.

Stephen Reed, while an engineer at Cycorp, designed a perceptual-motor front end for Cyc based on James Albus's Reference Model Architecture; the ensuing system, called CognitiveCyc, would have been the first full-fledged cognitive architecture based on Cyc, but was not implemented. Reed left Cycorp and is now building a system called Texai, which has many similarities to Cyc (and relies upon the OpenCyc knowledge base, a subset of Cyc's overall knowledge base), but incorporates a CognitiveCyc style cognitive architecture.

4.2.4 NARS

Pei Wang's NARS logic [Wan06] played a large role in the development of PLN, CogPrim's uncertain logic component, a relationship that is discussed in depth in [GMIH08] and won't be re-emphasized here. However, NARS is more than just an uncertain logic, it is also an overall cognitive architecture (which is centered on NARS logic, but also includes other aspects). CogPrime bears little relation to NARS except in the specific similarities between PLN logic and NARS logic, but, the other aspects of NARS are worth briefly recounting here.

NARS is formulated as a system for processing tasks, where a task consists of a question or a piece of new knowledge. The architecture is focused on declarative knowledge, but some pieces of knowledge may be associated with executable procedures, which allows NARS to carry out control activities (in roughly the same way that a Prolog program can).

At any given time a NARS system contains

- working memory: a small set of tasks which are active, kept for a short time, and closely related to new questions and new knowledge
- long-term memory: a huge set of knowledge which is passive, kept for a long time, and not necessarily related to current questions and knowledge

The working and long term memory spaces of NARS may each be thought of as a set of chunks, where each chunk consists of a set of tasks and a set of knowledge. NARS's basic cognitive process is:

1. choose a chunk
2. choose a task from that chunk
3. choose a piece of knowledge from that chunk
4. use the task and knowledge to do inference
5. send the new tasks to corresponding chunks

Depending on the nature of the task and knowledge, the inference involved may be one of the following:

- if the task is a question, and the knowledge happens to be an answer to the question, a copy of the knowledge is generated as a new task
- backward inference
- revision (merging two pieces of knowledge with the same form but different truth value)
- forward inference
- execution of a procedure associated with a piece of knowledge

Unlike many other systems, NARS doesn't decide what type of inference is used to process a task when the task is accepted, but works in a data-driven way – that is, it is the task and knowledge that dynamically determine what type of inference will be carried out

The "choice" processes mentioned above are done via assigning relative priorities to

- chunks (where they are called activity)
- tasks (where they are called urgency)
- knowledge (where they are called importance)

and then distributing the system's resources accordingly, based on a probabilistic algorithm. (It's interesting to note that while NARS uses probability theory as part of its control mechanism, the logic it uses to represent its own knowledge about the world is nonprobabilistic. This is considered conceptually consistent, in the context of NARS theory, because system control is viewed as a domain where the system's knowledge is more complete, thus more amenable to probabilistic reasoning.)

4.2.5 GLAIR and SNePS

Another logic-focused cognitive architecture, very different from NARS in detail, is Stuart Shapiro's GLAIR cognitive architecture, which is centered on the SNePS paraconsistent logic [SE07].

Like NARS, the core "cognitive loop" of GLAIR is based on reasoning: either thinking about some percept (e.g. linguistic input, or sense data from the virtual or physical world), or answering some question. This inference based cognition process is turned into an intelligent agent control process via coupling it with an acting component, which operates according to a set of policies, each one of which tells the system when to take certain internal or external actions (including internal reasoning actions) in response to its observed internal and external situation.

GLAIR contains multiple layers:

- the Knowledge Layer (KL), which contains the beliefs of the agent, and is where reasoning, planning, and act selection are performed

- the Perceptuo-Motor Layer (PML), which grounds the KL symbols in perceptual structures and subconscious actions, contains various registers for providing the agent's sense of situatedness in the environment, and handles translation and communication between the KL and the SAL.
- the Sensori-Actuator Layer (SAL), contains the controllers of the sensors and effectors of the hardware or software robot.

The logical Knowledge Layer incorporates multiple memory types using a common representation (including declarative, procedural, episodic, attentional and intentional knowledge, and meta-knowledge). To support this broad range of knowledge types, a broad range of logical inference mechanisms are used, so that the KL may be variously viewed as predicate logic based, frame based, semantic network based, or from other perspectives.

What makes GLAIR more robust than most logic based AI approaches is the novel paraconsistent logical formalism used in the knowledge base, which means (among other things) that uncertain, speculative or erroneous knowledge may exist in the system's memory without leading the system to create a broadly erroneous view of the world or carry out egregiously unintelligent actions. CogPrime is not thoroughly logic-focused like GLAIR is, but in its logical aspec it seeks a similar robustness through its use of PLN logic, which embodies properties related to paraconsistency.

Compared to CogPrime , we see that GLAIR has a similarly integrative approach, but that the integration of different sorts of cognition is done more strictly within the framework of logical knowledge representation.

4.3 Emergentist Cognitive Architectures

Another species of cognitive architecture expects abstract symbolic processing to emerge from lower-level "subsymbolic" dynamics, which sometimes (but not always) are designed to simulate neural networks or other aspects of human brain function. These architectures are typically strong at recognizing patterns in high-dimensional data, reinforcement learning and associative memory; but no one has yet shown how to achieve high-level functions such as abstract reasoning or complex language processing using a purely subsymbolic approach. A few of the more important subsymbolic, emergentist cognitive architectures are:

- **DeSTIN** [ARK09a, ARC09], which is part of CogPrime , may also be considered as an autonomous AGI architecture, in which case it is emergentist and contains mechanisms to encourage language, high-level reasoning and other abstract aspects of intelligent to emerge from hierarchical pattern recognition and related self-organizing network dynamics. In CogPrime DeSTIN is used as part of a hybrid architecture, which greatly reduces the reliance on DeSTIN's emergent properties.

- **Hierarchical Temporal Memory (HTM)** [HB06] is a hierarchical temporal pattern recognition architecture, presented as both an AI approach and a model of the cortex. So far it has been used exclusively for vision processing and we will discuss its shortcomings later in the context of our treatment of DeSTIN.
- **SAL** [JL08], based on the earlier and related **IBCA** (Integrated Biologically-based Cognitive Architecture) is a large-scale emergent architecture that seeks to model distributed information processing in the brain, especially the posterior and frontal cortex and the hippocampus. So far the architectures in this lineage have been used to simulate various human psychological and psycholinguistic behaviors, but hasn't been shown to give rise to higher-level behaviors like reasoning or subgoaling.
- **NOMAD** (Neurally Organized Mobile Adaptive Device) automata and its successors [KE06] are based on Edelman's "Neural Darwinism" model of the brain, and feature large numbers of simulated neurons evolving by natural selection into configurations that carry out sensorimotor and categorization tasks. The emergence of higher-level cognition from this approach seems rather unlikely.
- Ben Kuipers and his colleagues [**?**, MK08, MK09]have pursued an extremely innovative research program which combines qualitative reasoning and reinforcement learning to enable an intelligent agent to learn how to act, perceive and model the world. Kuipers' notion of "bootstrap learning" involves allowing the robot to learn almost *everything* about its world, including for instance the structure of 3D space and other things that humans and other animals obtain via their genetic endowments. Compared to Kuipers' approach, CogPrime falls in line with most other approaches which provide more "hard-wired" structure, following the analogy to biological organisms that are born with more innate biases.

There is also a set of emergentist architectures focused specifically on developmental robotics, which we will review below in a separate subsection, as all of these share certain common characteristics.

Our general perspective on the emergentist approach is that it is philosophically correct but currently pragmatically inadequate. Eventually, *some* emergentist approach could surely succeed at giving rise to humanlike general intelligence – the human brain, after all, is plainly an emergentist system. However, we currently lack understanding of how the brain gives rise to abstract reasoning and complex language, and none of the existing emergentist systems seem remotely capable of giving rise to such phenomena. It seems to us that the creation of a successful emergentist AGI will have to wait for either a detailed understanding of how the brain gives rise to abstract thought, or a much more thorough mathematical understanding of the dynamics of complex self-organizing systems.

The concept of cognitive synergy is more relevant to emergentist than to symbolic architectures. In a complex emergentist architecture with multiple

specialized components, much of the emergence is expected to arise via synergy between different richly interacting components. Symbolic systems, at least in the forms currently seen in the literature, seem less likely to give rise to cognitive synergy as their dynamics tend to be simpler. And hybrid systems, as we shall see, are somewhat diverse in this regard: some rely heavily on cognitive synergies and others consist of more loosely coupled components.

We now review the DeSTIN emergentist architecture in more detail, and then turn to the developmental robotics architectures.

4.3.1 DeSTIN: A Deep Reinforcement Learning Approach to AGI

The DeSTIN architecture, created by Itamar Arel and his colleagues, addresses the problem of general intelligence using hierarchical spatiotemporal networks designed to enable scalable perception, state inference and reinforcement-learning-guided action in real-world environments. DeSTIN has been developed with the plan of gradually extending it into a complete system for humanoid robot control, founded on the same qualitative information-processing principles as the human brain (though without striving for detailed biological realism). However, the practical work with DeSTIN to date has focused on visual and auditory processing; and in the context of the present proposal, the intention is to utilize DeSTIN for perception and actuation oriented processing, hybridizing it with CogPrime which will handle abstract cognition and language. Here we will discuss DeSTIN primarily in the perception context, only briefly mentioning the application to actuation which is conceptually similar.

In DeSTIN (see Figure 4.4), perception is carried out by a deep spatiotemporal inference network, which is connected to a similarly architected critic network that provides feedback on the inference network's performance, and an action network that controls actuators based on the activity in the inference network (Figure 4.5 depicts a standard action hierarchy, of which the hierarchy in DeSTIN is an example). The nodes in these networks perform probabilistic pattern recognition according to algorithms to be described below; and the nodes in each of the networks may receive states of nodes in the other networks as inputs, providing rich interconnectivity and synergetic dynamics.

4.3.1.1 Deep versus Shallow Learning for Perceptual Data Processing

The most critical feature of DeSTIN is its uniquely robust approach to modeling the world based on perceptual data. Mimicking the efficiency and ro-

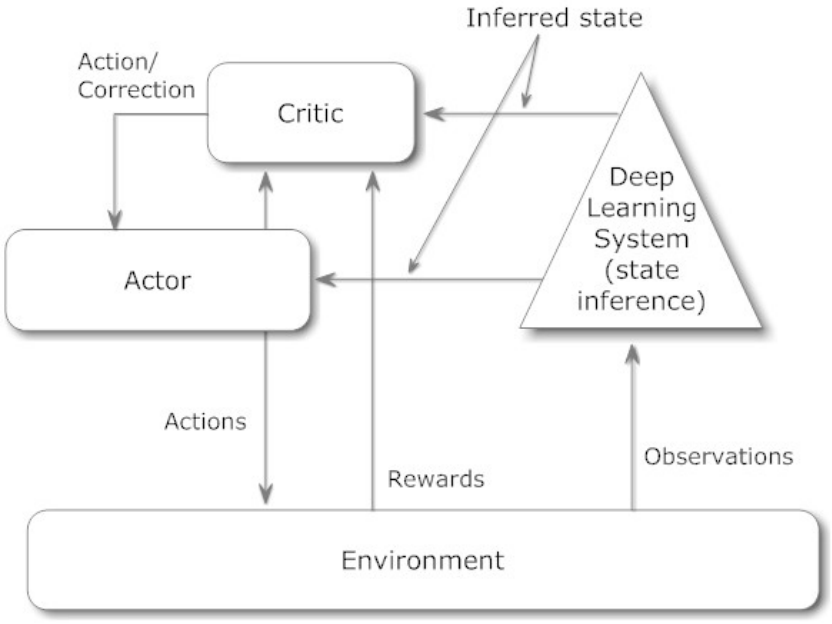


Fig. 4.4 High-level architecture of DeSTIN

bustness by which the human brain analyzes and represents information has been a core challenge in AI research for decades. For instance, humans are exposed to massive amounts of visual and auditory data every second of every day, and are somehow able to capture critical aspects of it in a way that allows for appropriate future recollection and action selection. For decades, it has been known that the brain is a massively parallel fabric, in which computation processes and memory storage are highly distributed. But massive parallelism is not in itself a solution – one also needs the right architecture; which DeSTIN provides, building on prior work in the area of deep learning.

Humanlike intelligence is heavily adapted to the physical environments in which humans evolved; and one key aspect of sensory data coming from our physical environments is its **hierarchical** structure. However, most machine learning and pattern recognition systems are "shallow" in structure, not explicitly incorporating the hierarchical structure of the world in their architecture. In the context of perceptual data processing, he practical result of this is the need to couple each shallow learner with a pre-processing stage, wherein high-dimensional sensory signals are reduced to a lower-dimension feature space that can be understood by the shallow learner. The hierarchical structure of the world is thus crudely captured in the hierarchy of "preprocessor plus shallow learner." In this sort of approach, much of the intelligence of the system shifts to the feature extraction process, which is often imperfect and always application-domain specific.

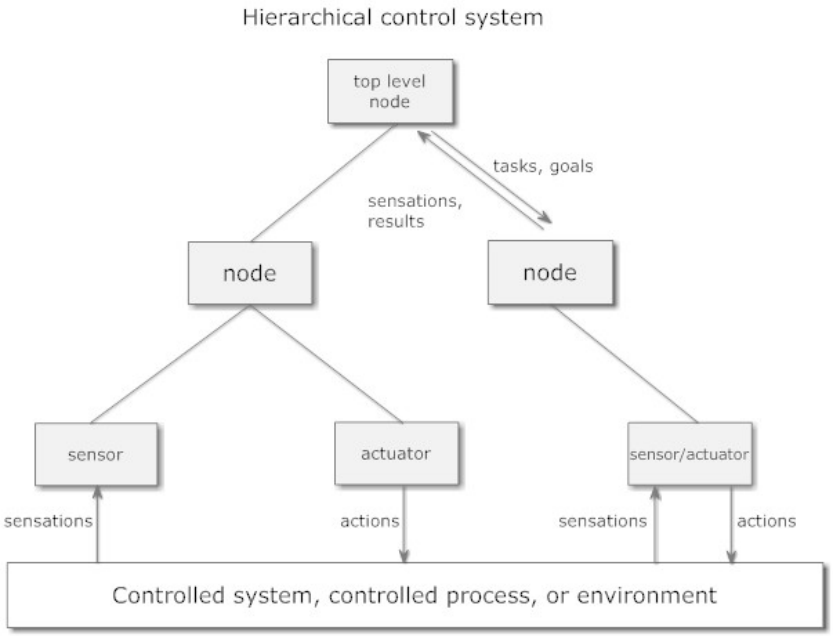


Fig. 4.5 A standard, general-purpose hierarchical control architecture. DeSTIN's control hierarchy exemplifies this architecture, with the difference lying mainly in the DeSTIN control hierarchy's tight integration with the state inference (perception) and critic (reinforcement) hierarchies.

Deep machine learning has emerged as a more promising framework for dealing with complex, high-dimensional real-world data. Deep learning systems possess a hierarchical structure that intrinsically biases them to recognize the hierarchical patterns present in real-world data. Thus, they hierarchically form a feature space that is driven by regularities in the observations, rather than by hand-crafted techniques. They also offer robustness to many of the distortions and transformations that characterize real-world signals, such as noise, displacement, scaling, etc.

Deep belief networks [HOT06] and Convolutional Neural Networks [LBDE90] have been demonstrated to successfully address pattern inference in high dimensional data (e.g. images). They owe their success to their underlying paradigm of partitioning large data structures into smaller, more manageable units, and discovering the dependencies that may or may not exist between such units. However, this paradigm has its limitations; for instance, these approaches do not represent temporal information with the same ease as spatial structure. Moreover, some key constraints are imposed on the learning schemes driving these architectures, namely the need for layer-by-layer training, and oftentimes pre-training. DeSTIN overcomes the limitations of

prior deep learning approaches to perception processing, and also extends beyond perception to action and reinforcement learning.

4.3.1.2 DeSTIN for Perception Processing

The hierarchical architecture of DeSTIN's spatiotemporal inference network comprises an arrangement into multiple layers of "nodes" comprising multiple instantiations of an identical cortical circuit. Each node corresponds to a particular spatiotemporal region, and uses a statistical learning algorithm to characterize the sequences of patterns that are presented to it by nodes in the layer beneath it. More specifically,

- At the very lowest layer of the hierarchy nodes receive as input raw data (e.g. pixels of an image) and continuously construct a belief state that attempts to characterize the sequences of patterns viewed.
- The second layer, and all those above it, receive as input the belief states of nodes at their corresponding lower layers, and attempt to construct belief states that capture regularities in their inputs.
- each node also receives as input the belief state of the node above it in the hierarchy (which constitutes "contextual" information)

More specifically, each of the DeSTIN nodes, referring to a specific space-time region, contains a set of state variables conceived as clusters, each corresponding to a set of previously-observed sequences of events. These clusters are characterized by centroids (and are hence assumed roughly spherical in shape), and each of them comprises a certain "spatiotemporal form" recognized by the system in that region. Each node then contains the predictive task of predicting the likelihood of a certain centroid being most apropos in the near future, based on the past history of observations in the node. This prediction may be done by simple probability tabulation, or via application of supervised learning algorithms such as recurrent neural networks. These clustering and prediction processes occur separately in each node, but the nodes are linked together via bidirectional dynamics: each node feeds input to its parents, and receives "advice" from its parents that is used to condition its probability calculations in a contextual way.

These processes are executed formally by the following basic belief update rule, which governs the learning process and is identical for every node in the architecture. The belief state is a probability mass function over the sequences of stimuli that the nodes learns to represent. Consequently, each node is allocated a predefined number of state variables each denoting a dynamic pattern, or sequence, that is autonomously learned. The DeSTIN update rule maps the current observation (o), belief state (b), and the belief state of a higher-layer node or context (c), to a new (updated) belief state (b'), such that

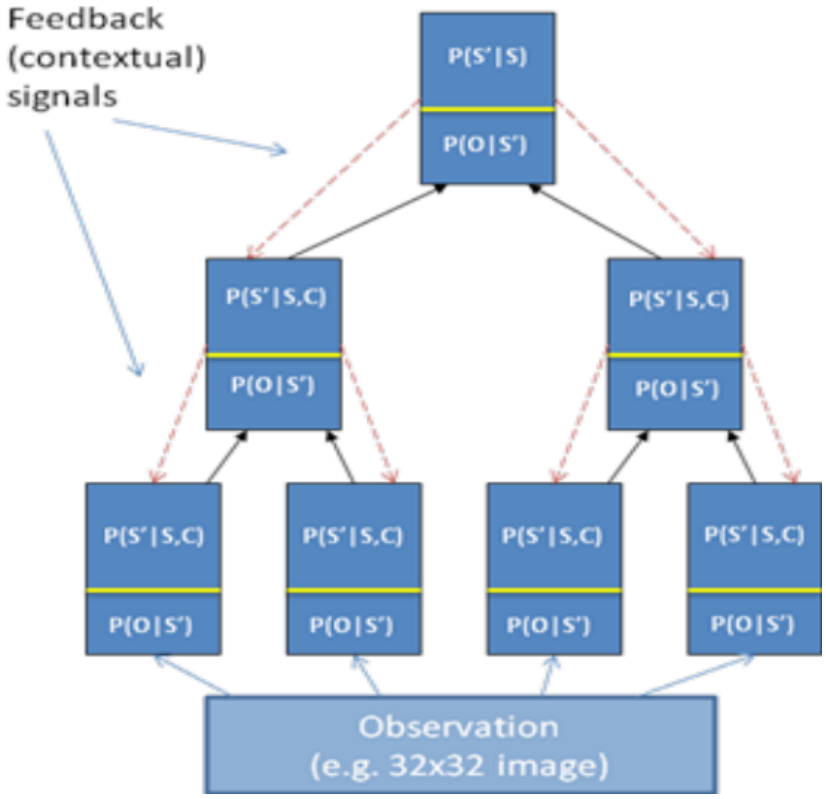


Fig. 4.6 Small-scale instantiation of the DeSTIN perceptual hierarchy. Each box represents a node, which corresponds to a spatiotemporal region (nodes higher in the hierarchy corresponding to larger regions). O denotes the current observation in the region, C is the state of the higher-layer node, and S and S‘ denote state variables pertaining to two subsequent time steps. In each node, a statistical learning algorithm is used to predict subsequent states based on prior states, current observations, and the state of the higher-layer node.

$$b'(s') = \Pr(s'|o,b,c) = \frac{\Pr(s' \cap o \cap b \cap c)}{\Pr(o \cap b \cap c)}, \tag{4.1}$$

alternatively expressed as

$$b'(s') = \frac{\Pr(o|s',b,c)\Pr(s'|b,c)\Pr(b,c)}{\Pr(o|b,c)\Pr(b,c)}. \tag{4.2}$$

Under the assumption that observations depend only on the true state, or $\Pr(o|s',b,c) = \Pr(o|s')$, we can further simplify the expression such that

$$b'(s') = \frac{\Pr(o|s')\Pr(s'|b,c)}{\Pr(o|b,c)}, \tag{4.3}$$

where $\Pr(s'|b,c) = \sum_{s \in S} \Pr(s'|s,c)\, b(s)$, yielding the belief update rule

$$b'(s') = \frac{\Pr(o|s') \sum_{s \in S} \Pr(s'|s,c)\, b(s)}{\sum_{s'' \in S} \Pr(o|s'') \sum_{s \in S} \Pr(s''|s,c)\, b(s)}, \tag{4.4}$$

where S denotes the sequence set (i.e. belief dimension) such that the denominator term is a normalization factor.

One interpretation of eq. (4.4) would be that the static pattern similarity metric, $\Pr(o|s')$, is modulated by a construct that reflects the system dynamics, $\Pr(s'|s,c)$. As such, the belief state inherently captures both spatial and temporal information. In our implementation, the belief state of the parent node, c, is chosen using the selection rule

$$c = \arg\max_{s} b_p(s), \tag{4.5}$$

where b_p is the belief distribution of the parent node.

A close look at eq. (4.4) reveals that there are two core constructs to be learned, $\Pr(o|s')$ and $\Pr(s'|s,c)$. In the current DeSTIN design, the former is learned via online clustering while the latter is learned based on experience by inductively learning a rule that predicts the next state s' given the prior state s and c.

The overall result is a robust framework that autonomously (i.e. with no human engineered pre-processing of any type) learns to represent complex data patterns, and thus serves the critical role of building and maintaining a model of the state of the world. In a vision processing context, for example, it allows for powerful unsupervised classification. If shown a variety of real-world scenes, it will automatically form internal structures corresponding to the various natural categories of objects shown in the scenes, such as trees, chairs, people, etc.; and also the various natural categories of events it sees, such as reaching, pointing, falling. And, as will be discussed below, it can use feedback from DeSTIN's action and critic networks to further shape its internal world-representation based on reinforcement signals.

Benefits of DeSTIN for Perception Processing

DeSTIN's perceptual network offers multiple key attributes that render it more powerful than other deep machine learning approaches to sensory data processing:

1. The belief space that is formed across the layers of the perceptual network inherently captures both *spatial and temporal regularities* in the data. Given that many applications require that temporal information

be discovered for robust inference, this is a key advantage over existing schemes.
2. Spatiotemporal regularities in the observations are captured in a coherent manner (rather than being represented via two separate mechanisms)
3. All processing is both top-down and bottom-up, and both hierarchical and heterarchical, based on nonlinear feedback connections directing activity and modulating learning in multiple directions through DeSTIN's cortical circuits
4. Support for multi-modal fusing is intrinsic within the framework, yielding a powerful state inference system for real-world, partially-observable settings.
5. Each node is identical, which makes it easy to map the design to massively parallel platforms, such as graphics processing units.

Points 2-4 in the above list describe how DeSTIN's perceptual network displays its own "cognitive synergy," in a way that fits naturally into the overall synergetic dynamics of the overall CogPrime architecture. Using this cognitive synergy, DeSTIN's perceptual network addresses a key aspect of general intelligence: the ability to robustly infer the state of the world, with which the system interacts, in an accurate and timely manner.

4.3.1.3 DeSTIN for Action and Control

DeSTIN's perceptual network performs unsupervised world-modeling, which is a critical aspect of intelligence but of course is not the whole story. DeSTIN's action network, coupled with the perceptual network, orchestrates actuator commands into complex movements, but also carries out other functions that are more cognitive in nature.

For instance, people learn to distinguish between cups and bowls in part via hearing other people describe some objects as cups and others as bowls. To emulate this kind of learning, DeSTIN's critic network provides positive or negative reinforcement signals based on whether the action network has correctly identified a given object as a cup or a bowl, and this signal then impacts the nodes in the action network. The critic network takes a simple external "degree or success or failure" signal and turns it into multiple reinforcement signals to be fed into the multiple layers of the action network. The result is that the action network self-organizes so as to include an implicit "cup versus bowl" classifier, whose inputs are the outputs of some of the nodes in the higher levels of the perceptual network. This classifier belongs in the action network because it is part of the procedure by which the DeSTIN system carries out the action of identifying an object as a cup or a bowl.

This example illustrates how the learning of complex concepts and procedures is divided fluidly between the perceptual network, which builds a model of the world in an unsupervised way, and the action network, which

learns how to respond to the world in a manner that will receive positive reinforcement from the critic network.

4.3.2 Developmental Robotics Architectures

A particular subset of emergentist cognitive architectures are sufficiently important that we consider them separately here: these are *developmental robotics* architectures, focused on controlling robots without significant "hard-wiring" of knowledge or capabilities, allowing robots to learn (and learn how to learn etc.) via their engagement with the world. A significant focus is often placed here on "intrinsic motivation," wherein the robot explores the world guided by internal goals like novelty or curiosity, forming a model of the world as it goes along, based on the modeling requirements implied by its goals. Many of the foundations of this research area were laid by Juergen Schmidhuber's work in the 1990s [Sch91, **?**, **?**, **?**], but now with more powerful computers and robots the area is leading to more impressive practical demonstrations.

We mention here a handful of the important initiatives in this area:

- Juyang Weng's **Dav** [HZT+02] and **SAIL** [WHZ+00] projects involve mobile robots that explore their environments autonomously, and learn to carry out simple tasks by building up their own world-representations through both unsupervised and teacher-driven processing of high-dimensional sensorimotor data. The underlying philosophy is based on human child development [WH06], the knowledge representations involved are neural network based, and a number of novel learning algorithms are involved, especially in the area of vision processing.
- **FLOWERS** [BO09], an initiative at the French research institute INRIA, led by Pierre-Yves Oudeyer, is also based on a principle of trying to reconstruct the processes of development of the human child's mind, spontaneously driven by intrinsic motivations. Kaplan [**?**] has taken this project in a directly closely related to the present one via the creation of a "robot playroom." Experiential language learning has also been a focus of the project [**?**], driven by innovations in speech understanding.
- **IM-CLEVER**[1], a new European project coordinated by Gianluca Baldassarre and conducted by a large team of researchers at different institutions, which is focused on creating software enabling an iCub [MSV+08] humanoid robot to explore the environment and learn to carry out human childlike behaviors based on its own intrinsic motivations. As this project is the closest to our own we will discuss it in more depth below.

Like CogPrime , IM-CLEVER is a humanoid robot intelligence architecture guided by intrinsic motivations, and using a hierarchical architectures for

[1] http://im-clever.noze.it/project/project-description

reinforcement learning and sensory abstraction. IM-CLEVER's motivational structure is based in part on Schmidhuber's information-theoretic model of curiosity [Sch06]; and CogPrime 's Psi-based motivational structure utilizes probabilistic measures of novelty, which are mathematically related to Schmidhuber's measures. On the other hand, IM-CLEVER's use of reinforcement learning follows Schmidhuber's earlier work RL for cognitive robotics [BS04, BZGS06], Barto's work on intrinsically motivated reinforcement learning [**?**, SBC05], and Lee's [LMC07b, LMC07a] work on developmental reinforcement learning; whereas CogPrime 's assemblage of learning algorithms is more diverse, including probabilistic logic, concept blending and other symbolic methods (in the OCP component) as well as more conventional reinforcement learning methods (in the DeSTIN component).

In many respects IM-CLEVER bears a moderately strong resemblance to DeSTIN, whose integration with CogPrime is discussed in Chapter 26 of Part 2 (although IM-CLEVER has much more focus on biological realism than DeSTIN). Apart from numerous technical differences, the really big distinction between IM-CLEVER and CogPrime is that in the latter we are proposing to hybridize a hierarchical-abstraction/reinforcement-learning system (such as DeSTIN) with a more abstract symbolic cognition engine that explicitly handles probabilistic logic and language. IM-CLEVER lacks the aspect of hybridization with a symbolic system, taking more of a pure emergentist strategy. Like DeSTIN considered as a standalone architecture IM-CLEVER does entail a high degree of cognitive synergy, between components dealing with perception, world-modeling, action and motivation. However, the "emergentist versus hybrid" is a large qualitative difference between the two approaches.

In all, while we largely agree with the philosophy underlying developmental robotics, our intuition is that the learning and representational mechanisms underlying the current systems in this area are probably not powerful enough to lead to human child level intelligence. We expect that these systems will develop interesting behaviors but fall short of robust preschool level competency, especially in areas like language and reasoning where symbolic systems have typically proved more effective. This intuition is what impels us to pursue a hybrid approach, such as CogPrime . But we do feel that eventually, once the mechanisms underlying brains are better understood and robotic bodies are richer in sensation and more adept in actuation, some sort of emergentist, developmental-robotics approach can be successful at creating humanlike, human-level AGI.

4.4 Hybrid Cognitive Architectures

In response to the complementary strengths and weaknesses of the symbolic and emergentist approaches, in recent years a number of researchers have

turned to integrative, hybrid architectures, which combine subsystems operating according to the two different paradigms. The combination may be done in many different ways, e.g. connection of a large symbolic subsystem with a large subsymbolic system, or the creation of a population of small agents each of which is both symbolic and subsymbolic in nature.

Nils Nilsson expressed the motivation for hybrid AGI systems very clearly in his article at the AI-50 conference (which celebrated the 50'th anniversary of the AI field) [**?**]. While affirming the value of the Physical Symbol System Hypothesis that underlies symbolic AI, he argues that "the PSSH explicitly assumes that, whenever necessary, symbols will be grounded in objects in the environment through the perceptual and effector capabilities of a physical symbol system." Thus, he continues,

"I grant the need for non-symbolic processes in some intelligent systems, but I think they supplement rather than replace symbol systems. I know of no examples of reasoning, understanding language, or generating complex plans that are best understood as being performed by systems using exclusively non-symbolic processes....

AI systems that achieve human-level intelligence will involve a combination of symbolic and non-symbolic processing."

A few of the more important hybrid cognitive architectures are:

- **CLARION** [SZ04] is a hybrid architecture that combines a symbolic component for reasoning on "explicit knowledge" with a connectionist component for managing "implicit knowledge." Learning of implicit knowledge may be done via neural net, reinforcement learning, or other methods. The integration of symbolic and subsymbolic methods is powerful, but a great deal is still missing such as episodic knowledge and learning and creativity. Learning in the symbolic and subsymbolic portions is carried out separately rather than dynamically coupled, minimizing "cognitive synergy" effects.
- **DUAL** [NK04] is the most impressive system to come out of Marvin Minsky's "Society of Mind" paradigm. It features a population of agents, each of which combines symbolic and connectionist representation, self-organizing to collectively carry out tasks such as perception, analogy and associative memory. The approach seems innovative and promising, but it is unclear how the approach will scale to high-dimensional data or complex reasoning problems due to the lack of a more structured high-level cognitive architecture.
- **LIDA** [BF09] is a comprehensive cognitive architecture heavily based on Bernard Baars' "Global Workspace Theory". It articulates a "cognitive cycle" integrating various forms of memory and intelligent processing in a single processing loop. The architecture ties in well with both neuroscience and cognitive psychology, but it deals most thoroughly with

"lower level" aspects of intelligence, handling more advanced aspects like language and reasoning only somewhat sketchily. There is a clear mapping between LIDA structures and processes and corresponding structures and processing in OCP; so that it's only a mild stretch to view CogPrime as an instantiation of the general LIDA approach that extends further both in the lower level (to enable robot action and sensation via DeSTIN) and the higher level (to enable advanced language and reasoning via OCP mechanisms that have no direct LIDA analogues).

- **MicroPsi** [Bac09] is an integrative architecture based on Dietrich Dorner's Psi model of motivation, emotion and intelligence. It has been tested on some practical control applications, and also on simulating artificial agents in a simple virtual world. MicroPsi's comprehensiveness and basis in neuroscience and psychology are impressive, but in the current version of MicroPsi, learning and reasoning are carried out by algorithms that seem unlikely to scale. OCP incorporates the Psi model for motivation and emotion, so that MicroPsi and CogPrime may be considered very closely related systems. But similar to LIDA, MicroPsi currently focuses on the "lower level" aspects of intelligence, not yet directly handling advanced processes like language and abstract reasoning.
- **PolyScheme** [Cas07] integrates multiple methods of representation, reasoning and inference schemes for general problem solving. Each Polyscheme "specialist" models a different aspect of the world using specific representation and inference techniques, interacting with other specialists and learning from them. Polyscheme has been used to model infant reasoning including object identity, events, causality, spatial relations. The integration of reasoning methods is powerful, but the overall cognitive architecture is simplistic compared to other systems and seems focused more on problem-solving than on the broader problem of intelligent agent control.
- **Shruti** [SA93] is a fascinating biologically-inspired model of human reflexive inference, represents in connectionist architecture relations, types, entities and causal rules using focal-clusters. However, much like Hofstadter's earlier Copycat architecture [Hof96], Shruti seems more interesting as a prototype exploration of ideas than as a practical AGI system; at least, after a significant time of development it has not proved significant effective in any applications
- James Albus's **4D/RCS** robotics architecture shares a great deal with some of the emergentist architectures discussed above, e.g. it has the same hierarchical pattern recognition structure as DeSTIN and HTM, and the same three cross-connected hierarchies as DeSTIN, and shares with the developmental robotics architectures a focus on real-time adaptation to the structure of the world. However, 4D/RCS is not foundationally learning-based but relies on hard-wired architecture and algorithms, intended to mimic the qualitative structure of relevant parts of the brain (and intended to be *augmented* by learning, which differentiates it from emergentist approaches.

As our own CogPrime approach is a hybrid architecture, it will come as no surprise that we believe several of the existing hybrid architectures are fundamentally going in the right direction. However, nearly all the existing hybrid architectures have severe shortcomings which we feel will prevent them from achieving robust humanlike AGI.

Many of the hybrid architectures are in essence "multiple, disparate algorithms carrying out separate functions, encapsulated in black boxes and communicating results with each other." For instance, PolyScheme, ACT-R and CLARION all display this "modularity" property to a significant extent. These architectures lack the rich, real-time interaction between the *internal dynamics* of various memory and learning processes that we believe is critical to achieving humanlike general intelligence using realistic computational resources. On the other hand, those architectures that feature richer integration – such as DUAL, Shruti, LIDA and MicroPsi – have the flaw of relying (at least in their current versions) on overly simplistic learning algorithms, which drastically limits their scalability.

It does seem plausible to us that some of these hybrid architectures could be dramatically extended or modified so as to produce humanlike general intelligence. For instance, one could replace LIDA's learning algorithms with others that interrelate with each other in a nuanced synergetic way; or one could replace MicroPsi's simple learning and reasoning methods with much more powerful and scalable ones acting on the same data structures. However, making these changes would dramatically alter the cognitive architectures in question on multiple levels.

4.4.1 Neural versus Symbolic; Global versus Local

The "symbolic versus emergentist" dichotomy that we have used to structure our review of cognitive architectures is not absolute nor fully precisely defined; it is more of a heuristic distinction. In this section, before plunging into the details of particular hybrid cognitive architectures, we review two other related dichotomies that are useful for understanding hybrid systems: *neural versus symbolic* systems, and *globalist versus localist* knowledge representation.

4.4.1.1 Neural-Symbolic Integration

The distinction between neural and symbolic systems has gotten fuzzier and fuzzier in recent years, with developments such as

- Logic-based systems being used to control embodied agents (hence using logical terms to deal with data that is apparently perception or actuation-

oriented in nature, rather than being symbolic in the semiotic sense), see [SS03b] and [GMIH08].
- Hybrid systems combining neural net and logical parts, or using logical or neural net components interchangeably in the same role [LAon].
- Neural net systems being used for strongly symbolic tasks such as automated grammar learning ([Elm91], [Elm91], plus more recent work.)

Figure 4.7 presents a schematic diagram of a generic neural-symbolic system, generalizing from [BH05], a paper that gives an elegant categorization of neural-symbolic AI systems. Figure 4.8 depicts several broad categories of neural-symbolic architecture.

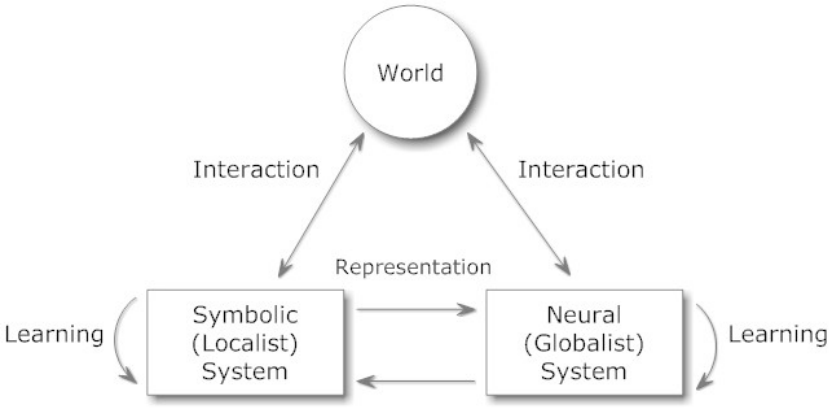


Fig. 4.7 Generic neural-symbolic architecture

Bader and Hitzler categorize neural-symbolic systems according to three orthogonal axes: interrelation, language and usage. "Language" refers to the type of language used in the symbolic component, which may be logical, automata-based, formal grammar-based, etc. "Usage" refers to the purpose to which the neural-symbolic interrelation is put. We tend to use "learning" as an encompassing term for all forms of ongoing knowledge-creation, whereas Bader and Hitzler distinguish learning from reasoning.

Of Bader and Hitzler's three axes the one that interests us most here is "interrelation", which refers to the way the neural and symbolic components of the architecture intersect with each other. They distinguish "hybrid" architectures which contain separate but equal, interacting neural and symbolic components; versus "integrative" architectures in which the symbolic component essentially rides piggyback on the neural component, extracting information from it and helping it carry out its learning, but playing a clearly derived and secondary role. We prefer Sun's (2001) term "monolithic" to Bader and Hitzler's "integrative" to describe this type of system, as the latter term seems best preserved in its broader meaning.

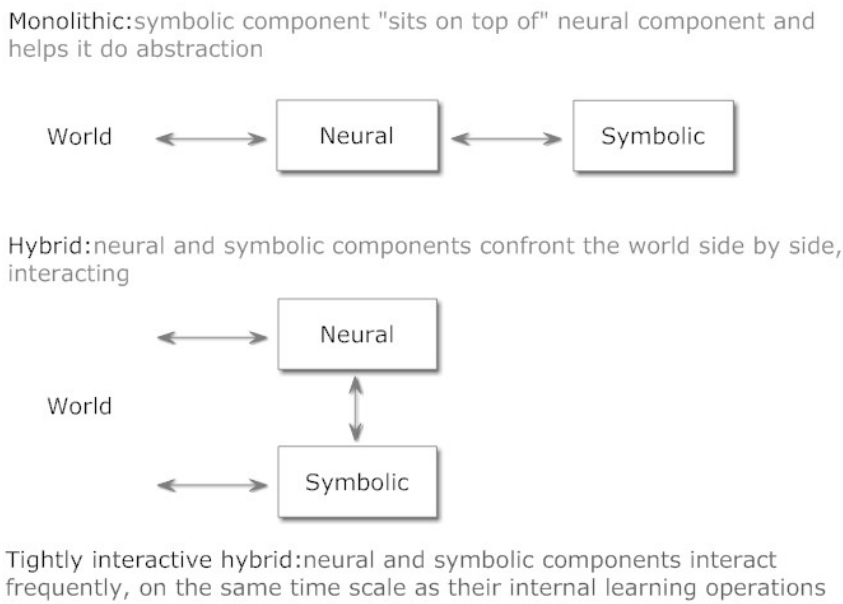


Fig. 4.8 Broad categories of neural-symbolic architecture

Within the scope of hybrid neural-symbolic systems, there is another axis which Bader and Hitzler do not focus on, because the main interest of their review is in monolithic systems. We call this axis "interactivity," and what we are referring to is the frequency of high-information-content, high-influence interaction between the neural and symbolic components in the hybrid system. In a low-interaction hybrid system, the neural and symbolic components don't exchange large amounts of mutually influential information all that frequently, and basically act like independent system components that do their learning/reasoning/thinking periodically send each other their conclusions. In some cases, interaction may be asymmetric: one component may frequently send a lot of influential information to the other, but vice versa. However, our hypothesis is that the most capable neural-symbolic systems are going to be the symmetrically highly interactive ones.

In a symmetric high-interaction hybrid neural-symbolic system, the neural and symbolic components exchange influential information sufficiently frequently that each one plays a major role in the other one's learning/reasoning/thinking processes. Thus, the learning processes of each component must be considered as part of the overall dynamic of the hybrid system. The two components aren't just feeding their outputs to each other as inputs, they're mutually guiding each others' internal processing.

One can make a speculative argument for the relevance of this kind of architecture to neuroscience. It seems plausible that this kind of neural-symbolic system roughly emulates the kind of interaction that exists between

the brain's neural subsystems implementing localist symbolic processing, and the brain's neural subsystems implementing globalist, classically "connectionist" processing. It seems most likely that, in the brain, symbolic functionality emerges from an underlying layer of neural dynamics. However, it is also reasonable to conjecture that this symbolic functionality is confined to a functionally distinct subsystem of the brain, which then interacts with other subsystems in the brain much in the manner that the symbolic and neural components of a symmetric high-interaction neural-symbolic system interact.

Neuroscience speculations aside, however, our key conjecture regarding neural-symbolic integration is that this sort of neural-symbolic system presents a promising direction for artificial general intelligence research. In Chapter 26 of Volume 2 we will give a more concrete idea of what a symmetric high-interaction hybrid neural-symbolic architecture might look like, exploring the potential for this sort of hybridization between the OpenCogPrime AGI architecture (which is heavily symbolic in nature) and hierarchical attractor neural net based architectures such as DeSTIN.

4.5 Globalist versus Localist Representations

Another interesting distinction, related to but different from "symbolic versus emergentist" and "neural versus symbolic", may be drawn between cognitive systems (or subsystems) where memory is essentially **global**, and those where memory is essentially **local**. In this section we will pursue this distinction in various guises, along with the less familiar notion of **glocal memory**.

This globalist/localist distinction is most easily conceptualized by reference to memories corresponding to categories of entities or events in an external environment. In an AI system that has an internal notion of "activation" – i.e. in which some of its internal elements are more active than others, at any given point in time – one can define the *internal image* of an external event or entity as the fuzzy set of internal elements that tend to be active when that event or entity is presented to the system's sensors. If one has a particular set S of external entities or events of interest, then, the *degree of memory localization* of such an AI system relative to S may be conceived as the percentage of the system's internal elements that have a high degree of membership in the internal image of an average element of S.

Of course, this characterization of localization has its limitations, such as the possibility of ambiguity regarding what are the "system elements" of a given AI system; and the exclusive focus on internal images of external phenomena rather than representation of internal abstract concepts. However, our goal here is not to formulate an ultimate, rigorous and thorough ontology of memory systems, but only to pose a "rough and ready" categorization so as to properly frame our discussion of some specific AGI issues relevant to Cog-

Prime . Clearly the ideas pursued here will benefit from further theoretical exploration and elaboration.

In this sense, a Hopfield neural net [Ami89] would be considered "globalist" since it has a low degree of memory localization (most internal images heavily involve a large number of system elements); whereas Cyc would be considered "localist" as it has a very high degree of memory localization (most internal images are heavily focused on a small set of system elements).

However, although Hopfield nets and Cyc form handy examples, the "globalist vs. localist" distinction as described above is not identical to the "neural vs. symbolic" distinction. For it is in principle quite possible to create localist systems using formal neurons, and also to create globalist systems using formal logic. And "globalist-localist" is not quite identical to "symbolic vs emergentist" either, because the latter is about coordinated system dynamics and behavior not just about knowledge representation. CogPrime combines both symbolic and (loosely) neural representations, and also combines globalist and localist representations in a way that we will call "glocal" and analyze more deeply in Chapter 13; but there are many other ways these various properties could be manifested by AI systems. Rigorously studying the corpus of existing (or hypothetical!) cognitive architectures using these ideas would be a large task, which we do not undertake here.

In the next sections we review several hybrid architectures in more detail, focusing most deeply on LIDA and MicroPsi which have been directly inspirational for CogPrime .

4.5.1 CLARION

Ron Sun's CLARION architecture (see Figure 4.9) is interesting in its combination of symbolic and neural aspects – a combination that is used in a sophisticated way to embody the distinction and interaction between implicit and explicit mental processes. From a CLARION perspective, architectures like Soar and ACT-R are severely limited in that they deal only with explicit knowledge and associated learning processes.

CLARION consists of a number of distinct subsystems, each of which contains a dual representational structure, including a "rules and chunks" symbolic knowledge store somewhat similar to ACT-R, and a neural net knowledge store embodying implicit knowledge. The main subsystems are:

- An action-centered subsystem to control actions;
- A non-action-centered subsystem to maintain general knowledge;
- A motivational subsystem to provide underlying motivations for perception, action, and cognition;
- A meta-cognitive subsystem to monitor, direct, and modify the operations of all the other subsystems.

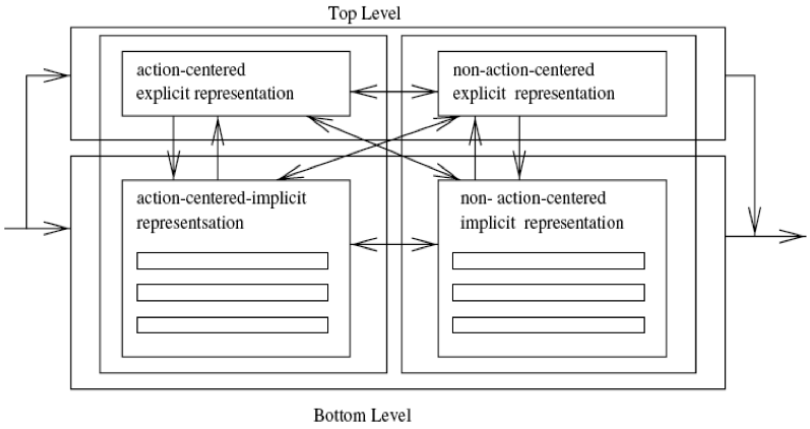


Fig. 4.9 The CLARION cognitive architecture.

4.5.2 The Society of Mind and the Emotion Machine

In his influential but controversial book *The Society of Mind* [Min88], Marvin Minsky described a model of human intelligence as something that is built up from the interactions of numerous simple agents. He spells out in great detail how various particular cognitive functions may be achieved via agents and their interactions. He leaves no room for any central algorithms or structures of thought, famously arguing: "What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity, not from any single, perfect principle."

This perspective was extended in the more recent work *The Emotion Machine* [Min07], where Minsky argued that emotions are "ways to think" evolved to handle different "problem types" that exist in the world. The brain is posited to have rule-based mechanisms (selectors) that turns on emotions to deal with various problems.

Overall, both of these works serve better as works of speculative cognitive science than as works of AI or cognitive architecture per se. As neurologist Richard Restak said in his review of *Emotion Machine*, "Minsky does a marvelous job parsing other complicated mental activities into simpler elements. ... But he is less effective in relating these emotional functions to what's going on in the brain." As Restak did not add, he is also not so effective at relating these emotional functions to straightforwardly implementable algorithms or data structures.

Push Singh, in his PhD thesis and followup work [SBC05], did the best job so far of creating a concrete AI design based on Minsky's ideas. While Singh's system was certainly interesting, it was also noteworthy for its lack of any learning mechanisms, and its exclusive focus on explicit rather than

implicit knowledge. Due to Singh's tragic death, his work was never brought anywhere near completion. It seems fair to say that there has not yet been a serious cognitive architecture posed based closely on Minsky's ideas.

4.5.3 DUAL

The closest thing to a Minsky-ish cognitive architecture is probably DUAL, which takes the Society of Mind concept and adds to it a number of other interesting ideas. DUAL integrates symbolic and connectionist approaches at a deeper level than CLARION, and has been used to model various cognitive functions such as perception, analogy and judgment. Computations in DUAL emerge from the self-organized interaction of many micro-agents, each of which is a hybrid symbolic/connectionist device. Each DUAL agent plays the role of a neural network node, with an activation level and activation spreading dynamics; but also plays the role of a symbol, manipulated using formal rules. The agents exchange messages and activation via links that can be learned and modified, and they form coalitions which collectively represent concepts, episodes, and facts.

The structure of the model is sketchily depicted in Figure 4.10, which covers the application of DUAL to a toy environment called TextWorld. The visual input corresponding to a stimulus is presented on a two-dimensional visual array representing the front end of the system. Perceptual primitives like blobs and terminations are immediately generated by cheap parallel computations. Attention is controlled at each time by an object which allocates it selectively to some area of the stimulus. A detailed symbolic representation is constructed for this area which tends to fade away as attention is withdrawn from it and allocated to another one. Categorization of visual memory contents takes place by retrieving object and scene categories from DUAL's semantic memory and mapping them onto current visual memory representations.

In principle the DUAL framework seems quite powerful; using the language of CogPrime , however, it seems to us that the learning mechanisms of DUAL have not been formulated in such a way as to give rise to powerful, scalable cognitive synergy. It would likely be possible to create very powerful AGI systems within DUAL, and perhaps some very CogPrime -like systems as well. But the systems that have been created or designed for use within DUAL so far seem not to be that powerful in their potential or scope.

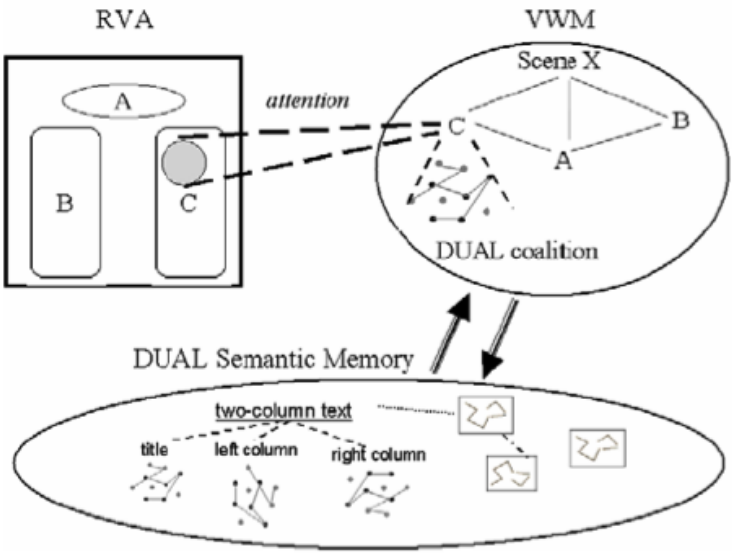


Fig. 4.10 The three main components of the DUAL model: the retinotopic visual array (RVA), the visual working memory (VWM) and DUAL's semantic memory. Attention is allocated to an area of the visual array by the object in VWM controlling attention, while scene and object categories corresponding to the contents of VWM are retrieved from the semantic memory.

4.5.4 4D/RCS

In a rather different direction, James Albus, while at the National Bureau of Standards, developed a very thorough and impressive architecture for intelligent robotics called 4D/RCS, which was implemented in a number of machines including unmanned automated vehicles. This architecture lacks critical aspects of intelligence such as learning and creativity, but combines perception, action, planning and world-modeling in a highly effective and tightly-integrated fashion.

The architecture has three hierarchies of memory/processing units: one for perception, one for action and one for modeling and guidance. Each unit has a certain spatiotemporal scope, and (except for the lowest level) supervenes over children whose spatiotemporal scope is a subset of its own. The action hierarchy takes care of decomposing tasks into subtasks; whereas the sensation hierarchy takes care of grouping signals into entities and events. The modeling/guidance hierarchy mediates interactions between perception and action based on its understanding of the world and the system's goals.

In his book [AM01] Albus describes methods for extending 4D/RCS into a complete cognitive architecture, but these extensions have not been elaborated in full detail nor implemented.

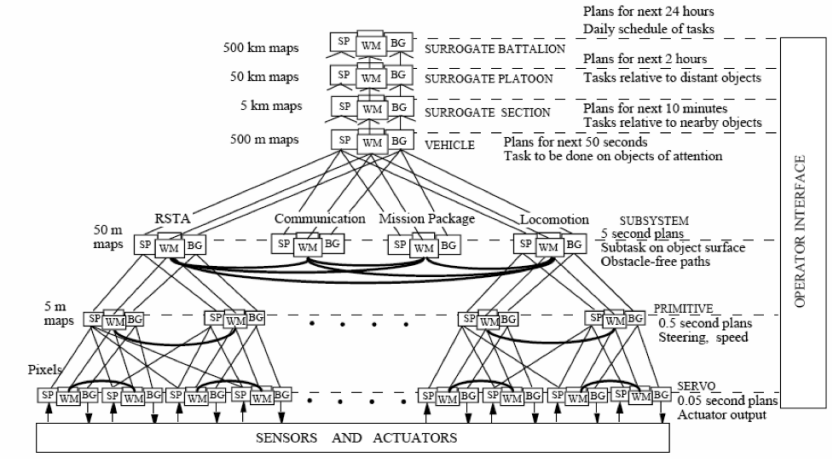


Fig. 4.11 Albus's 4D-RCS architecture for a single vehicle

4.5.5 PolyScheme

Nick Cassimatis's PolyScheme architecture [Cas07] shares with GLAIR the use of multiple logical reasoning methods on a common knowledge store. While its underlying ideas are quite general, currently PolyScheme is being developed in the context of the "object tracking" domain (construed very broadly). As a logic framework PolyScheme is fairly conventional (unlike GLAIR or NARS with their novel underlying formalisms), but PolyScheme has some unique conceptual aspects, for instance its connection with Cassimatis's theory of mind, which holds that the same core set of logical concepts and relationships underlies both language and physical reasoning [Cas04]. This ties in with the use of a common knowledge store for multiple cognitive processes; for instance it suggests that

- the same core relationships can be used for physical reasoning and parsing, but that each of these domains may involve some additional relationships.
- language processing may be done via physical-reasoning-based cognitive processes, plus the additional activity of some language-specific processes

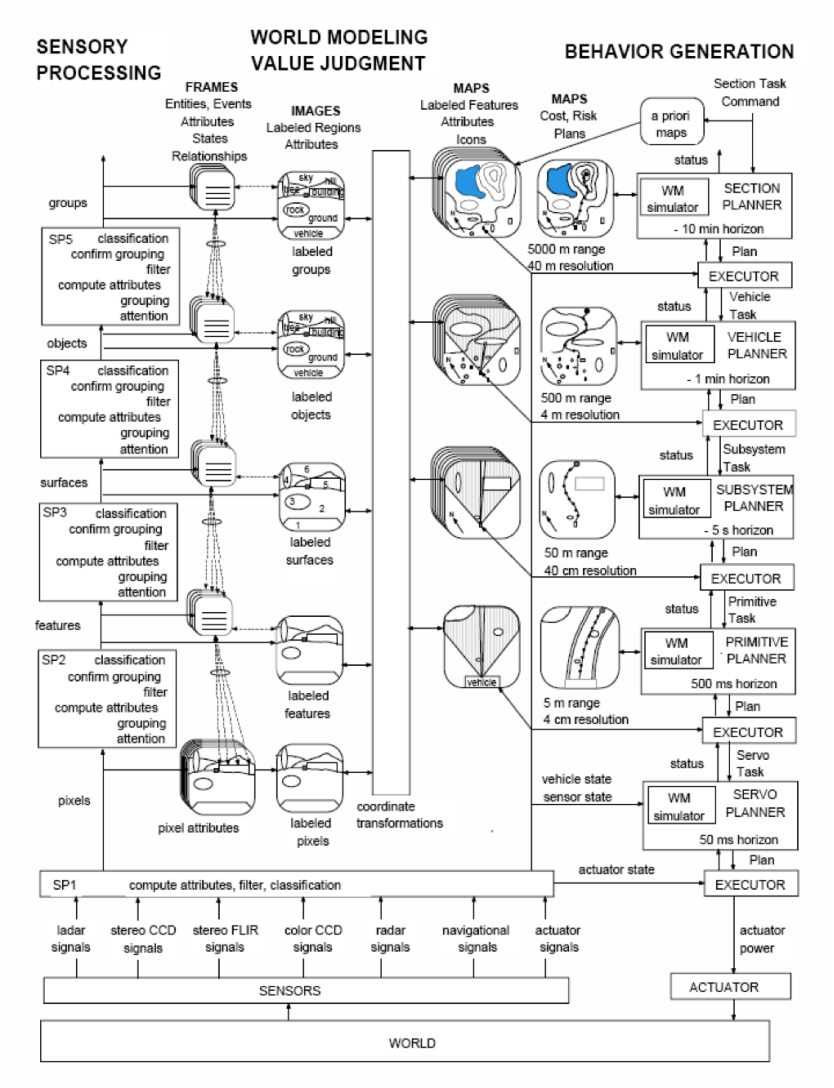


Fig. 4.12 Albus's perceptual, motor and modeling hierarchies

4.5.6 Joshua Blue

Sam Adams and his colleagues at IBM have created a cognitive architecture called Joshua Blue [AABL02], which has some significant similarities to CogPrime . Similar to our current research direction with CogPrime , Joshua Blue was created with loose emulation of child cognitive development in mind; and, also similar to CogPrime , it features a number of cognitive

processes acting on a common neural-symbolic knowledge store. The specific cognitive processes involved in Joshua Blue and CogPrime are not particularly similar, however. At time of writing (2009) Joshua Blue is not under active development and has not been for some time; however, the project may be reanimated in future.

Joshua Blue's core knowledge representation is a semantic network of nodes connected by links along which activation spreads. Although many of the nodes have specific semantic referents, as in a classical semantic net, the spread of activation through the network is designed to lead to the emergence of "assemblies" (which could also be thought of as dynamical attractors) in a manner more similar to an attractor neural network.

A major difference from typical semantic or neural network models is the central role that affect plays in the system's dynamics. The weights of the links in the knowledge base are adjusted dynamically based on the emotional context – a very direct way of ensuring that cognitive processes and mental representations are continuously influenced by affect. Qualitatively, this mimics the way that particular emotions in the human brain correlate with the dissemination throughout the brain of particular neurotransmitters, which then affect synaptic activity.

A result of this architecture is that in Joshua Blue, emotion directs attention in a very direct way: affective weighting is important in determining which associated objects will become part of the focus of attention, or will be retained from memory. A notable similarity between CogPrime and Joshua Blue is that in both systems, nodes are assigned two quantitative attention values, one governing allocation of current system resources (mainly processor time; this is CogPrime 's ShortTermImportance) and one governing the long-term allocation of memory (CogPrime 's LongTermImportance).

The concrete work done with Joshua Blue involved using it to control a simple agent in a simulated world, with the goal that via human interaction, the agent would develop a complex and humanlike emotional and motivational structure from its simple in-built emotions and drives, and would then develop complex cognitive capabilities as part of this development process.

4.5.7 LIDA

The LIDA architecture developed by Stan Franklin and his colleagues [BF09] is based on the concept of the "cognitive cycle" - a notion that is important to nearly every BICA and also to the brain, but that plays a particularly central role in LIDA. As Franklin says, "as a matter of principle, every autonomous agent, be it human, animal, or artificial, must frequently sample (sense) its environment, process (make sense of) this input, and select an appropriate response (action). The agent's "life" can be viewed as consisting of a continual sequence of iterations of these cognitive cycles. Such cycles constitute the

indivisible elements of attention, the least sensing and acting to which we can attend. A cognitive cycle can be thought of as a moment of cognition, a cognitive "moment.""

4.5.8 The Global Workspace

LIDA is heavily based on the "global workspace" concept developed by Bernard Baars. As this concept is also directly relevant to CogPrime it is worth briefly describing here.

In essence Baars' Global Workspace Theory (GWT) is a particular hypothesis about how working memory works and the role it plays in the mind. Baars conceives working memory as the "inner domain in which we can rehearse telephone numbers to ourselves or, more interestingly, in which we carry on the narrative of our lives. It is usually thought to include inner speech and visual imagery." Baars uses the term "consciousness" to refer to the contents of working memory – a theoretical commitment that is not part of the CogPrime design. In this section we will use the term "consciousness" in Baars' way, but not throughout the rest of the book.

Baars conceives working memory and consciousness in terms of a "theater metaphor" – according to which, in the "theater of consciousness" a "spotlight of selective attention" shines a bright spot on stage. The bright spot reveals the global workspace – the contents of consciousness, which may be metaphorically considered as a group of actors moving in and out of consciousness, making speeches or interacting with each other. The unconscious is represented by the audience watching the play ... and there is also a role for the director (the mind's executive processes) behind the scenes, along with a variety of helpers like stage hands, script writers, scene designers, etc.

GWT describes a fleeting memory with a duration of a few seconds. This is much shorter than the 10-30 seconds of classical working memory – according to GWT there is a very brief "cognitive cycle" in which the global workspace is refreshed, and the time period an item remains in working memory generally spans a large number of these elementary "refresh" actions. GWT contents are proposed to correspond to what we are conscious of, and are said to be broadcast to a multitude of unconscious cognitive brain processes. Unconscious processes, operating in parallel, can form coalitions which can act as input processes to the global workspace. Each unconscious process is viewed as relating to certain goals, and seeking to get involved with coalitions that will get enough importance to become part of the global workspace – because once they're in the global workspace they'll be allowed to broadcast out across the mind as a whole, which include broadcasting to the internal and external actuators that allow the mind to do things. Getting into the global workspace is a process's best shot at achieving its goals.

Obviously, the theater metaphor used to describe the GWT is evocative but limited; for instance, the unconscious in the mind does a lot more than the audience in a theater. The unconscious comes up with complex creative ideas sometimes, which feed into consciousness – almost as if the audience is also the scriptwriter. Baars' theory, with its understanding of unconscious dynamics in terms of coalition-building, fails to describe the subtle dynamics occurring within the various forms of long-term memory, which result in subtle nonlinear interactions between long term memory and working memory. But nevertheless, GWT successfully models a number of characteristics of consciousness, including its role in handling novel situations, its limited capacity, its sequential nature, and its ability to trigger a vast range of unconscious brain processes. It is the framework on which LIDA's theory of the cognitive cycle is built.

4.5.9 The LIDA Cognitive Cycle

The simplest cognitive cycle is that of an animal, which senses the world, compares sensation to memory, and chooses an action, all in one fluid subjective moment. But the same cognitive cycle structure/process applies to higher-level cognitive processes as well. The LIDA architecture is based on the LIDA model of the cognitive cycle, which posits a particular structure underlying the cognitive cycle that possess the generality to encompass both simple and complex cognitive moments.

The LIDA cognitive cycle itself is a theoretical construct that can be implemented in many ways, and indeed other BICAs like CogPrime and Psi also manifest the LIDA cognitive cycle in their dynamics, though utilizing different particular structures to do so.

Figure 4.13 shows the cycle pictorially, starting in the upper left corner and proceeding clockwise. At the start of a cycle, the LIDA agent perceives its current situation and allocates attention differentially to various parts of it. It then broadcasts information about the most important parts (which constitute the agent's consciousness), and this information gets features extracted from it, when then get passed along to episodic and semantic memory, that interact in the "global workspace" to create a model fo the agent's current situation. This model then, in interaction with procedural memory, enables the agent to choose an appropriate action and execute it - the critical "action-selection" phase!

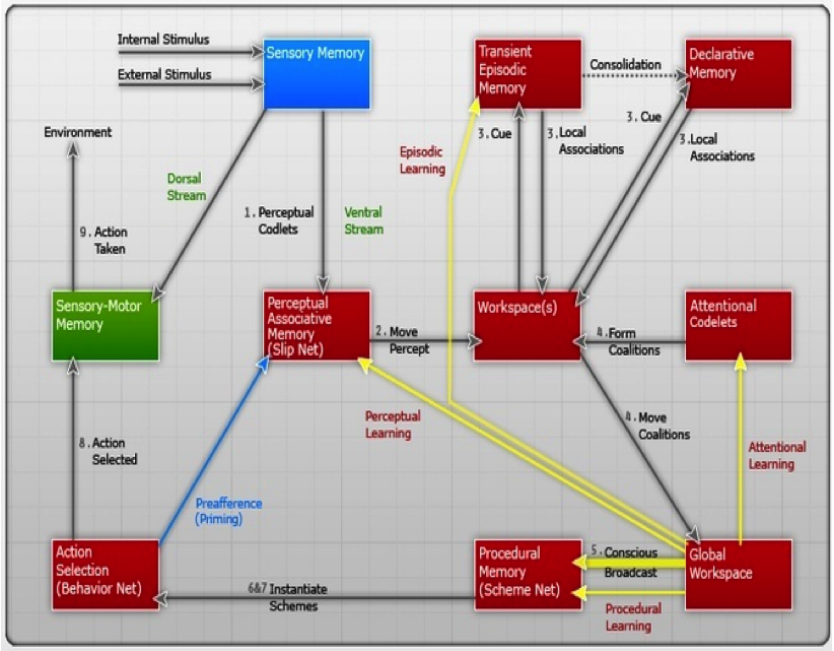


Fig. 4.13 The LIDA Cognitive Cycle

The LIDA Cognitive Cycle in More Depth

[2]

We now run through the cognitive cycle in more detail. It begins with sensory stimuli from the agent's external internal environment. Low-level feature detectors in sensory memory begin the process of making sense of the incoming stimuli. These low-level features are passed to perceptual memory where higher-level features, objects, categories, relations, actions, situations, etc. are recognized. These recognized entities, called percepts, are passed to the workspace, where a model of the agent's current situation is assembled.

Workspace structures serve as cues to the two forms of episodic memory, yielding both short and long term remembered local associations. In addition to the current percept, the workspace contains recent percepts that haven't yet decayed away, and the agent's model of the then-current situation previously assembled from them. The model of the agent's current situation is updated from the previous model using the remaining percepts and associations. This updating process will typically require looking back to perceptual memory and even to sensory memory, to enable the understanding of relations and situations. This assembled new model constitutes the agent's

[2] This section paraphrases heavily from [Fra06]

understanding of its current situation within its world. Via constructing the model, the agent has made sense of the incoming stimuli.

Now attention allocation comes into play, because a real agent lacks the computational resources to work with all parts of its world-model with maximal mental focus. Portions of the model compete for attention. These competing portions take the form of (potentially overlapping) coalitions of structures comprising parts the model. Once one such coalition wins the competition, the agent has decided what to focus its attention on.

And now comes the purpose of all this processing: to help the agent to decide what to do next. The winning coalition passes to the global workspace, the namesake of Global Workspace Theory, from which it is broadcast globally. Though the contents of this conscious broadcast are available globally, the primary recipient is procedural memory, which stores templates of possible actions including their context and possible results.

Procedural memory also stores an activation value for each such template – a value that attempts to measure the likelihood of an action taken within its context producing the expected result. It's worth noting that LIDA makes a rather specific assumption here. LIDA's "activation" values are like the probabilistic truth values of the implications in CogPrime 's $Context \wedge Procedure \rightarrow Goal$ triples. However, in CogPrime this probability is not the same as the ShortTermImportance "attention value" associated with the Implication link representing that implication. Here LIDA merges together two concepts that in CogPrime are separate.

Templates whose contexts intersect sufficiently with the contents of the conscious broadcast instantiate copies of themselves with their variables specified to the current situation. These instantiations are passed to the action selection mechanism, which chooses a single action from these instantiations and those remaining from previous cycles. The chosen action then goes to sensorimotor memory, where it picks up the appropriate algorithm by which it is then executed. The action so taken affects the environment, and the cycle is complete.

The LIDA model hypothesizes that all human cognitive processing is via a continuing iteration of such cognitive cycles. It acknowledges that other cognitive processes may also occur, refining and building on the knowledge used in the cognitive cycle (for instance, the cognitive cycle itself doesn't mention abstract reasoning or creativity). But the idea is that these other processes occur in the context of the cognitive cycle, which is the main loop driving the internal and external activities of the organism.

4.5.9.1 Avoiding Combinatorial Explosion via Adaptive Attention Allocation

LIDA avoids combinatorial explosions in its inference processes via two methods, both of which are also important in CogPrime :

- combining reasoning via association with reasoning via deduction
- foundational use of uncertainty in reasoning

One can create an analogy between LIDA's workspace structures and codelets and a logic-based architecture's assertions and functions. However, LIDA's codelets only operate on the structures that are active in the workspace during any given cycle. This includes recent perceptions, their closest matches in other types of memory, and structures recently created by other codelets. The results with the highest estimate of success, i.e. activation, will then be selected.

Uncertainty plays a role in LIDA's reasoning in several ways, most notably through the base activation of its behavior codelets, which depend on the model's estimated probability of the codelet's success if triggered. LIDA observes the results of its behaviors and updates the base activation of the responsible codelets dynamically.

We note that for this kind of uncertain inference/activation interplay to scale well, some level of cognitive synergy must be present; and based on our understanding of LIDA it is not clear to us whether the particular inference and association algorithms used in LIDA possess the requisite synergy.

4.5.9.2 LIDA versus CogPrime

The LIDA cognitive cycle, broadly construed, exists in CogPrime as in other cognitive architectures. To see how it suffices to map the key LIDA structures into corresponding CogPrime structures, as is done in Table 4.1. Of course this table does not cover all CogPrime processes, as LIDA does not constitute a thorough explanation of CogPrime structure and dynamics. And in most cases the corresponding CogPrime and LIDA processes don't work in exactly the same way; for instance, as noted above, LIDA's action selection relies solely on LIDA's "activation" values, whereas CogPrime 's action selection process is more complex, relying on aspects of CogPrime that lack LIDA analogues.

4.5.10 Psi and MicroPsi

We have saved for last the architecture that has the most in common with CogPrime : Joscha Bach's MicroPsi architecture, closely based on Dietrich Dorner's Psi theory. CogPrime has borrowed substantially from Psi in its handling of emotion and motivation; but Psi also has other aspects that differ considerably from CogPrime . Here we will focus more heavily on the points of overlap, but will mention the key points of difference as well.

The overall Psi cognitive architecture, which is centered on the Psi model of the motivational system, is roughly depicted in Figure 4.14.

LIDA	CogPrime
Declarative memory	Atomspace
attentional codelets	Schema that adjust importance of Atoms explicitly
coalitions	maps
global workspace	attentional focus
behavior codelets	schema
procedural memory (scheme net)	procedures in ProcedureRepository; and network of SchemaNodes in the Atomspace
action selection (behavior net)	propagation of STICurrency from goals to actions, and action selection process
transient episodic memory	perceptual atoms entering AT with high STI, which rapidly decreases in most cases
local workspaces	bubbles of interlinked Atoms with moderate importance, focused on by a subset of MindAgents (defined in Chapter 19 of Part 2) for a period of time
perceptual associative memory	HebbianLinks in the AT
sensory memory	spaceserver/timeserver, plus auxiliary stores for other senses
sensorimotor memory	Atoms storing record of actions taken, linked in with Atoms indexed in sensory memory

Table 4.1 CogPrime Analogues of Key LIDA Features

Psi's motivational system begins with **Demands**, which are the basic factors that motivate the agent. For an animal these would include things like food, water, sex, novelty, socialization, protection of one's children, and so forth. For an intelligent robot they might include things like electrical power, novelty, certainty, socialization, well-being of others and mental growth.

Psi also specifies two fairly abstract demands and posits them as psychologically fundamental (see Figure 4.15):

- **competence**, the effectiveness of the agent at fulfilling its Urges
- **certainty**, the confidence of the agent's knowledge

Each demand is assumed to come with a certain "target level" or "target range" (and these may fluctuate over time, or may change as a system matures and develops). An **Urge** is said to develop when a demand deviates from its target range: the urge then seeks to return the demand to its target range. For instance, in an animal-like agent the demand related to food is more clearly described as "fullness," and there is a target range indicating that the agent is neither too hungry nor too full of food. If the agent's fullness deviates from this range, an Urge to return the demand to its target range arises. Similarly, if an agent's novelty deviates from its target range, this means the agent's life has gotten either too boring or too disconcertingly weird, and the agent gets an Urge for either more interesting activities (in the case of below-range novelty) or more familiar ones (in the case of above-range novelty).

There is also a primitive notion of **Pleasure** (and its opposite, displeasure), which is considered as different from the complex emotion of "happi-

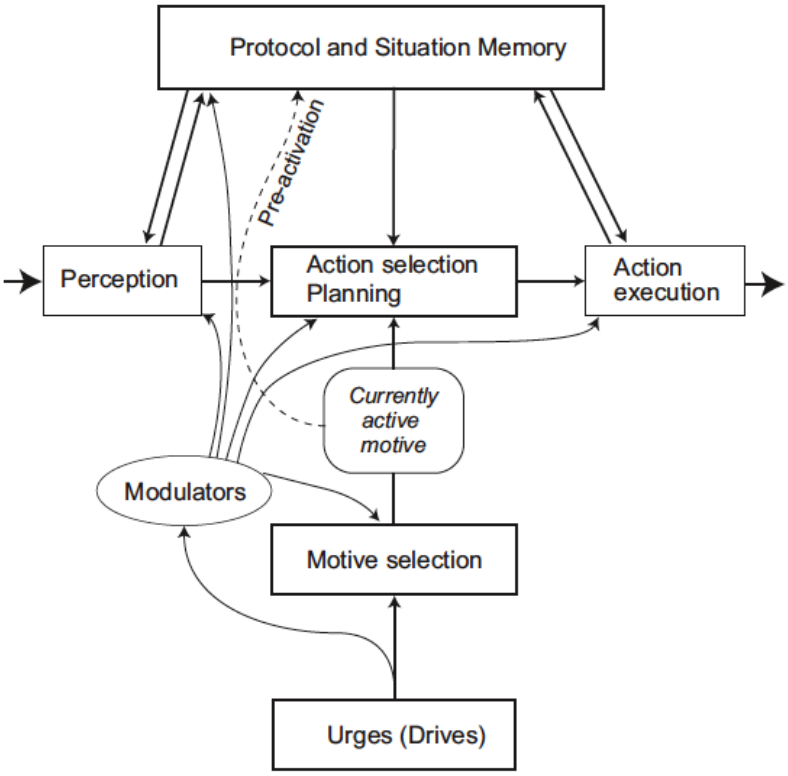


Fig. 4.14 High-Level Architecture of the Psi Model

ness." Pleasure is understood as associated with Urges: pleasure occurs when an Urge is (at least partially) satisfied, whereas displeasure occurs when an urge gets increasingly severe. The degree to which an Urge is satisfied is not necessarily defined instantaneously; it may be defined, for instance, as a time-decaying weighted average of the proximity of the demand to its target range over the recent past.

So, for instance if an agent is bored and gets a lot of novel stimulation, then it experiences some pleasure. If it's bored and then the monotony of its stimulation gets even more extreme, then it experiences some displeasure.

Note that, according to this relatively simplistic approach, any decrease in the amount of dissatisfaction causes some pleasure; whereas if everything always continues within its acceptable range, there isn't any pleasure. This may seem a little counterintuitive, but it's important to understand that these simple definitions of "pleasure" and "displeasure" are not intended to fully capture the natural language concepts associated with those words. The natural language terms are used here simply as heuristics to convey the

general character of the processes involved. These are very low level processes whose analogues in human experience are largely below the conscious level.

A **Goal** is considered as a statement that the system may strive to make true at some future time. A **Motive** is an $(urge, goal)$ pair, consisting of a goal whose satisfaction is predicted to imply the satisfaction of some urge. In fact one may consider Urges as top-level goals, and the agent's other goals as their subgoals.

In Psi an agent has one "ruling motive" at any point in time, but this seems an oversimplification more applicable to simple animals than to human-like or other advanced AI systems. In general one may think of different motives having different weights indicating the amount of resources that will be spent on pursuing them .

Emotions in Psi are considered as complex systemic response-patterns rather than explicitly constructed entities. An emotion is the set of mental entities activated in response to a certain set of urges. Dorner conceived theories about how various common emotions emerge from the dynamics of urges and motives as described in the Psi model. "Intentions" are also considered as composite entities: an intention at a given point in time consists of the active motives, together with their related goals, behavior programs and so forth.

The basic logic of action in Psi is carried out by "triples" that are very similar to CogPrime 's $Context \wedge Procedure \rightarrow Goal$ triples. However, an important role is played by four **modulators** that control how the processes of perception, cognition and action selection are regulated at a given time:

- *activation*, which determines the degree to which the agent is focused on rapid, intensive activity versus reflective, cognitive activity
- *resolution level*, which determines how accurately the system tries to perceive the world
- *certainty*, which determines how hard the system tries to achieve definite, certain knowledge
- *selection threshold*, which determines how willing the system is to change its choice of which goals to focus on

These modulators characterize the system's emotional and cognitive state at a very abstract level; they are not emotions per se, but they have a large effect on the agent's emotions. Their intended interaction is depicted in Figure 4.15.

4.5.11 The Emergence of Emotion in the Psi Model

We now briefly review the specifics of how Psi models the emergence of emotion. The basic idea is to define a small set of **proto-emotional dimensions**

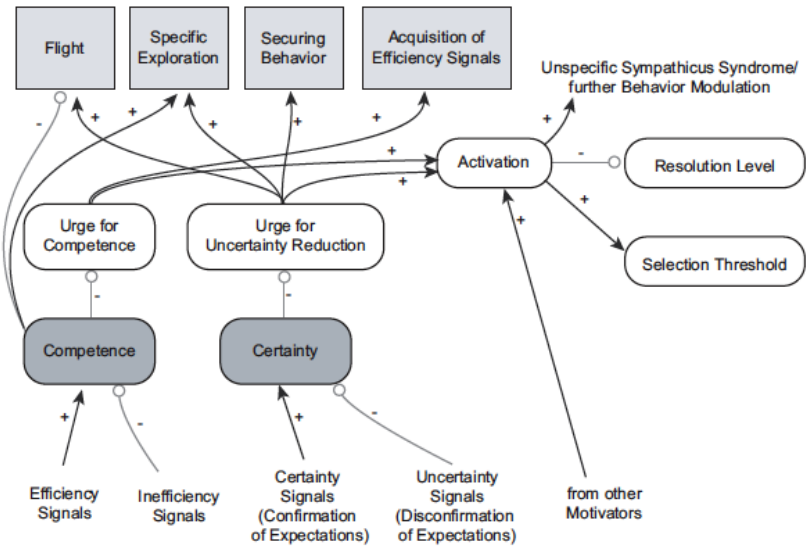


Fig. 4.15 Primary Interrelationships Between Psi Modulators

in terms of basic Urges and modulators. Then, emotions are identified with regions in the space spanned by these dimensions.

The simplest approach uses a six-dimensional continuous space:

1. pleasure
2. arousal
3. resolution level
4. selection threshold (i.e. degree of dominance of the leading motive)
5. level of background checks (the rate of the securing behavior)
6. level of goal-directed behavior

Figure 4.16 shows how the latter 5 of these dimensions are derived from underlying urges and modulators. Note that these dimensions are not orthogonal; for instance resolution is mainly inversely related to arousal. Additional dimensions are also discussed, for instance it is postulated that to deal with social emotions one may wish to introduce two more demands corresponding to inner and outer obedience to social norms, and then define dimensions in terms of these.

Specific emotions are then characterized in terms of these dimensions. According to [Bac09], for instance, "Anger ... is characterized by high arousal, low resolution, strong motive dominance, few background checks and strong goal-orientedness; sadness by low arousal, high resolution, strong dominance, few background-checks and low goal-orientedness."

I'm a bit skeptical of the contention that these dimensions *fully* characterize the relevant emotions. Anger for instance seems to have some particular

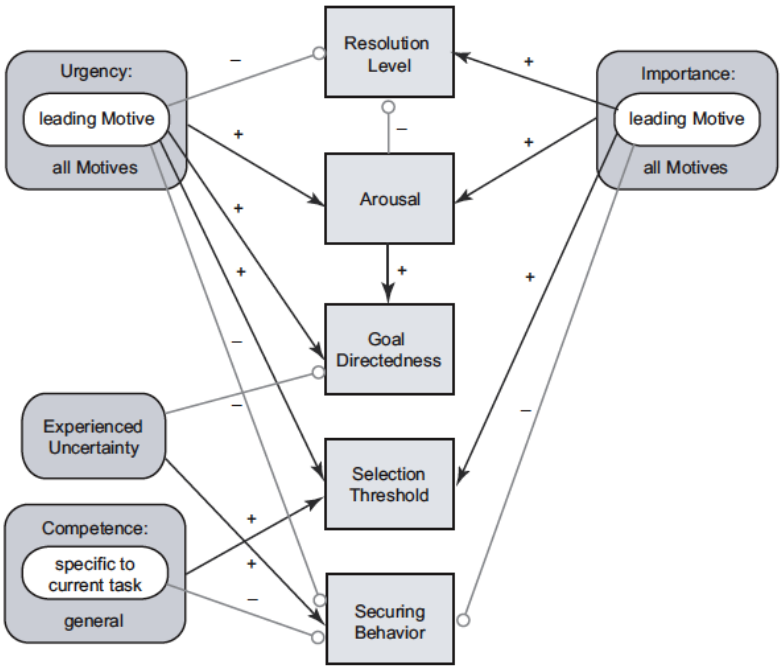


Fig. 4.16 Five Proto-Emotional Dimensions Implicit in the Psi Model

characteristics not implied by the above list of dimensional values. The list of dimensional values associated with anger doesn't tell us that an angry person is more likely to punch someone than to bounce up and down, for example. However, it does seem that the dimensional values associated with an emotion are informative about the emotion, so that positioning an emotion on the given dimensions tells one a lot.

4.5.12 Knowledge Representation, Action Selection and Planning in Psi

In addition to the basic motivation/emotion architecture of Psi, which has been adopted (with some minor changes) for use in CogPrime , Psi has a number of other aspects that are somewhat different from their CogPrime analogues.

First of all, on the micro level, Psi represents knowledge using structures called "quads." Each quad is a cluster of 5 neurons containing a core neuron, and four other neurons representing before/after and part-of/has-part rela-

tionships in regard to that core neuron. Quads are naturally assembled into spatiotemporal hierarchies, though they are not required to form part of such a structure.

Psi stores knowledge using quads arranged in three networks, which are conceptually similar to the networks in Albus's 4D/RCS and Arel's DeSTIN architectures:

- A sensory network, which stores declarative knowledge: schemas representing images, objects, events and situations as hierarchical structures.
- A motor network, which contains procedural knowledge by way of hierarchical behavior programs
- A motivational network handling demands

Perception in Psi, which is centered in the sensory network, follows principles similar to DeSTIN (which are shared also by other systems), for instance the principle of *perception as prediction*. Psi's "HyPercept" mechanism performs hypothesis-based perception: it attempts to predict what is there to be perceived and then attempts to verify these predictions using sensation and memory. Furthermore HyPercept is intimately coupled with actions in the external world, according to the concept of "Neisser's perceptual cycle," the cycle between exploration and representation of reality. Perceptually acquired information is translated into schemas capable of guiding behaviors, and these are enacted (sometimes affecting the world in significant ways) and in the process used to guide further perception. Imaginary perceptions are handled via a "mental stage" analogous to CogPrime 's internal simulation world.

Action selection in Psi works based on what are called "triplets," each of which consists of

- a sensor schema (pre-conditions, "condition schema"; like CogPrime 's "context")
- a subsequent motor schema (action, effector; like CogPrime 's "procedure")
- a final sensor schema (post-conditions, expectations; like an CogPrime predicate or goal)

What distinguishes these triplets from classic production rules as used in (say) Soar and ACT-R is that the triplets may be partial (some of the three elements may be missing) and may be uncertain. However, there seems no fundamental difference between these triplets and CogPrime 's concept/procedure/goal triplets, at a high level; the difference lies in the underlying knowledge representation used for the schemata, and the probabilistic logic used to represent the implication

The work of figuring out what schema to execute to achieve the chosen goal in the current context is done in Psi using a combination of processes called the "Rasmussen ladder" (named after Danish psychologist Jens Rasmussen). The Rasmussen ladder describes the organization of action as a

movement between the stages of skill-based behavior, rule-based behavior and knowledge-based behavior, as follows:

- If a given task amounts to a trained routine, an automatism or skill is activated; it can usually be executed without conscious attention and deliberative control.
- If there is no automatism available, a course of action might be derived from rules; before a known set of strategies can be applied, the situation has to be analyzed and the strategies have to be adapted.
- In those cases where the known strategies are not applicable, a way of combining the available manipulations (operators) into reaching a given goal has to be explored at first. This stage usually requires a recomposition of behaviors, that is, a planning process.

The planning algorithm used in the Psi and MicroPsi implementations is a fairly simple hill-climbing planner. While it's hypothesized that a more complex planner may be needed for advanced intelligence, part of the Psi theory is the hypothesis that most real-life planning an organism needs to do is fairly simple, once the organism has the right perceptual representations and goals.

4.5.13 Psi versus CogPrime

On a high level, the similarities between Psi and CogPrime are quite strong:

- interlinked declarative, procedural and intentional knowledge structures, represented using neural-symbolic methods (though, the knowledge structures have somewhat different high-level structures and low-level representational mechanisms in the two systems)
- perception via prediction and perception/action integration
- action selection via triplets that resemble uncertain, potentially partial production rules
- similar motivation/emotion framework, since CogPrime incorporates a variant of Psi for this

On the nitty-gritty level there are many differences between the systems, but on the big-picture level the *main* difference lies in the way the cognitive synergy principle is pursued in the two different approaches. Psi and MicroPsi rely on very simple learning algorithms that are closely tied to the "quad" neurosymbolic knowledge representation, and hence interoperate in a fairly natural way without need for subtle methods of "synergy engineering." CogPrime uses much more diverse and sophisticated learning algorithms which thus require more sophisticated methods of interoperation in order to achieve cognitive synergy.

Chapter 5
A Generic Architecture of Human-Like Cognition

5.1 Introduction

When writing the first draft of this book, some years ago, we had the idea to explain CogPrime by aligning its various structures and processes with the ones in the "standard architecture diagram" of the human mind. After a bit of investigation, though, we gradually came to the realization that no such thing existed. There was no standard flowchart or other sort of diagram explaining the modern consensus on how human thought works. Many such diagrams existed, but each one seemed to represent some particular focus or theory, rather than an overall integrative understanding.

Since there are multiple opinions regarding nearly every aspect of human intelligence, it would be difficult to get two cognitive scientists to fully agree on every aspect of an overall human cognitive architecture diagram. Prior attempts to outline detailed mind architectures have tended to follow highly specific theories of intelligence, and hence have attracted only moderate interest from researchers not adhering to those theories. An example is Minsky's work presented in *The Emotion Machine* [Min07], which arguably does constitute an architecture diagram for the human mind, but which is only loosely grounded in current empirical knowledge and stands more as a representation of Minsky's own intuitive understanding.

But nevertheless, it seemed to us that a reasonable attempt at an integrative, relatively theory-neutral "human cognitive architecture diagram" would be better than nothing. So naturally, we took it on ourselves to create such a diagram. This chapter is the result – it draws on the thinking of a number of cognitive science and AGI researchers, integrating their perspectives in a coherent, overall architecture diagram for human, and human-like, general intelligence. The specific architecture diagram of CogPrime , given in Chapter 6 below, may then be understood as a particular instantiation of this generic architecture diagram of human-like cognition.

There is no getting around the fact that, to a certain extent, the diagram presented here reflects our particular understanding of how the mind works. However, it was intentionally constructed with the goal of *not* being just an abstracted version of the CogPrime architecture diagram! It does not reflect our own idiosyncratic understanding of human intelligence, as much as a combination of understandings previously presented by multiple researchers (including ourselves), arranged according to our own taste in a manner we find conceptually coherent. With this in mind, we call it the "Integrative Human-Like Cognitive Architecture Diagram," or for short "the integrative diagram." We have made an effort to ensure that as many pieces of the integrative diagram as possible are well grounded in psychological and even neuroscientific data, rather than mainly embodying speculative notions; however, given the current state of knowledge, this could not be done to a complete extent, and there is still some speculation involved here and there.

While based on understandings of human intelligence, the integrative diagram is intended to serve as an architectural outline for human-like general intelligence more broadly. For example, CogPrime is explicitly not intended as a precise emulation of human intelligence, and does many things quite differently than the human mind, yet can still fairly straightforwardly be mapped into the integrative diagram.

The integrative diagram focuses on *structure*, but this should not be taken to represent a valuation of structure over dynamics in our approach to intelligence. Following chapters treat various dynamical phenomena in depth.

5.2 Key Ingredients of the Integrative Human-Like Cognitive Architecture Diagram

The main ingredients we've used in assembling the integrative diagram are as follows:

- Our own views on the various types of memory critical for human-like cognition, and the need for tight, "synergetic" interactions between the cognitive processes focused on these
- Aaron Sloman's high-level architecture diagram of human intelligence [Slo01], drawn from his CogAff archtiecture, which is strikes me as a particularly clear embodiment of "modern common sense" regarding the overall architecture of the human mind. We have added only a couple items to Sloman's high-level diagram, which we felt deserved an explicit high-level role that he did not give them: emotion, language and reinforcement.
- The LIDA architecture diagram presented by Stan Franklin and Bernard Baars [BF09]. We think LIDA is an excellent model of working memory and what Sloman calls "reactive processes", with well-researched ground-

ing in the psychology and neuroscience literature. We have adapted the LIDA diagram only very slightly for use here, changing some of the terminology on the arrows, and indicating where parts of the LIDA diagram indicate processes elaborated in more detail elsewhere in the integrative diagram.

- The architecture diagram of the Psi model of motivated cognition, presented by Joscha Bach in [Bac09] based on prior work by Dietrich Dorner [Dör02]. This diagram is presented without significant modification; however it should be noted that Bach and Dorner present this diagram in the context of larger and richer cognitive models, the other aspects of which are not all incorporated in the integrative diagram.
- James Albus's three-hierarchy model of intelligence [AM01], involving coupled perception, action and reinforcement hierarchies. Albus's model, utilized in the creation of intelligent unmanned automated vehicles, is a crisp embodiment of many ideas emergent from the field of intelligent control systems.
- Deep learning networks as a model of perception (and action and reinforcement learning), as embodied for example in the work of Itamar Arel [ARC09] and Jeff Hawkins [HB06]. The integrative diagram adopts this as the basic model of the perception and action subsystems of human intelligence. Language understanding and generation are also modeled according to this paradigm.

One possible negative reaction to the integrative diagram might be to say that it's a kind of Frankenstein monster diagram, piecing together aspects of different theories in a way that violates the theoretical notions underlying all of them! For example, the integrative diagram takes LIDA as a model of working memory and reactive processing, but from the papers on LIDA it's unclear whether the creators of LIDA construe it more broadly than that. The deep learning community tends to believe that the architecture of current deep learning networks, in itself, is close to sufficient for human-level general intelligence – whereas the integrative diagram appropriates the ideas from this community mainly for handling perception, action and language. Etc.

On the other hand, in a more positive perspective, one could view the integrative diagram as consistent with LIDA, but merely providing much more detail on some of the boxes in the LIDA diagram (e.g. dealing with perception and long-term memory). And one could view the integrative diagram as consistent with consistent with the deep learning paradigm – via viewing it, not as a description of components to be explicitly implemented in an AGI system, but rather as a description of the key structures and processes that must emerge in deep learning network, based on its engagement with the world, in order for it to achieve human-like general intelligence.

Our own view, underlying the creation of the integrative diagram, is that different communities of cognitive science researchers have focused on different aspects of intelligence, and have thus each created models that are more fully fleshed out in some aspects than others. But these various models all

link together fairly cleanly, which is not surprising as they are all grounded in the same data regarding human intelligence. Many judgment calls must be made in fusing multiple models in the way that the integrative diagram does, but we feel these can be made without violating the spirit of the component models. In assembling the integrative diagram, we have made these judgment calls as best we can, but we're well aware that different judgments would also be feasible and defensible. Revisions are likely as time goes on, not only due to new data about human intelligence but also to evolution of understanding regarding the best approach to model integration.

Another possible argument against the ideas presented here is that there's nothing new – all the ingredients presented have been given before elsewhere. To this our retort is to quote Pascal: "Let no one say that I have said nothing new ... the arrangement of the subject is new." The various architecture diagrams incorporated into the integrative diagram are either extremely high level (Sloman's diagram) or focus primarily on one aspect of intelligence, treating the others very concisely by summarizing large networks of distinction structures and processes in small boxes. The integrative diagram seeks to cover all aspects of human-like intelligence at a roughly equal granularity – a different arrangement.

This kind of high-level diagramming exercise is not precise enough, nor dynamics-focused enough, to serve as a guide for creating human-level or more advanced AGI. But it can be a useful tool for explaining and interpreting a concrete AGI design, such as CogPrime .

5.3 An Architecture Diagram for Human-Like General Intelligence

The integrative diagram is presented here in a series of five Figures.

Figure 5.1 gives a high-level breakdown into components, based on Sloman's high-level cognitive-architectural sketch [Slo01]. This diagram represents, roughly speaking, "modern common sense" about how a human-like mind is architected. The separation between structures and processes, embodied in having separate boxes for Working Memory vs. Reactive Processes, and for Long Term Memory vs. Deliberative Processes, could be viewed as somewhat artificial, since in the human brain and most AGI architectures, memory and processing are closely integrated. However, the tradition in cognitive psychology is to separate out Working Memory and Long Term Memory from the cognitive processes acting thereupon, so we have adhered to that convention. The other changes from Sloman's diagram are the explicit inclusion of language, representing the hypothesis that language processing is handled in a somewhat special way in the human brain; and the inclusion of a reinforcement component parallel to the perception and action hierarchies, as inspired by intelligent control systems theory (e.g. Albus as mentioned

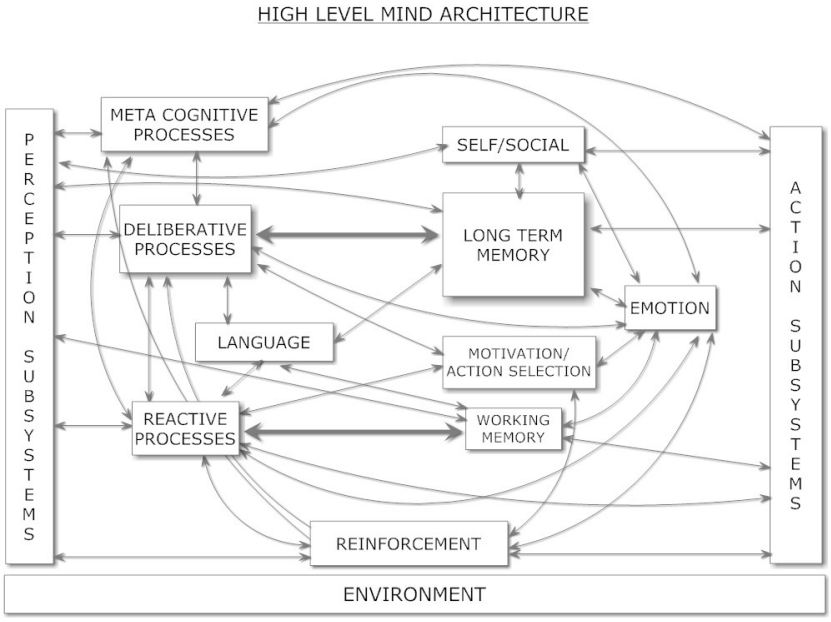


Fig. 5.1 High-Level Architecture of a Human-Like Mind

above) and deep learning theory. Of course Sloman's high level diagram in its original form is intended as inclusive of language and reinforcement, but we felt it made sense to give them more emphasis.

Figure 5.2, modeling working memory and reactive processing, is essentially the LIDA diagram as given in prior papers by Stan Franklin, Bernard Baars and colleagues [BF09]. The boxes in the upper left corner of the LIDA diagram pertain to sensory and motor processing, which LIDA does not handle in detail, and which are modeled more carefully by deep learning theory. The bottom left corner box refers to action selection, which in the integrative diagram is modeled in more detail by Psi. The top right corner box refers to Long-Term Memory, which the integrative diagram models in more detail as a synergetic multi-memory system (Figure 5.4).

The original LIDA diagram refers to various "codelets", a key concept in LIDA theory. We have replaced "attention codelets" here with "attention flow", a more generic term. We suggest one can think of an attention codelet as a piece of information that it's currently pertinent to pay attention to a certain collection of items together.

Figure 5.3, modeling motivation and action selection, is a lightly modified version of the Psi diagram from Joscha Bach's book *Principles of Synthetic Intelligence* [Bac09]. The main difference from Psi is that in the integrative diagram the Psi motivated action framework is embedded in a larger, more complex cognitive model. Psi comes with its own theory of working and long-

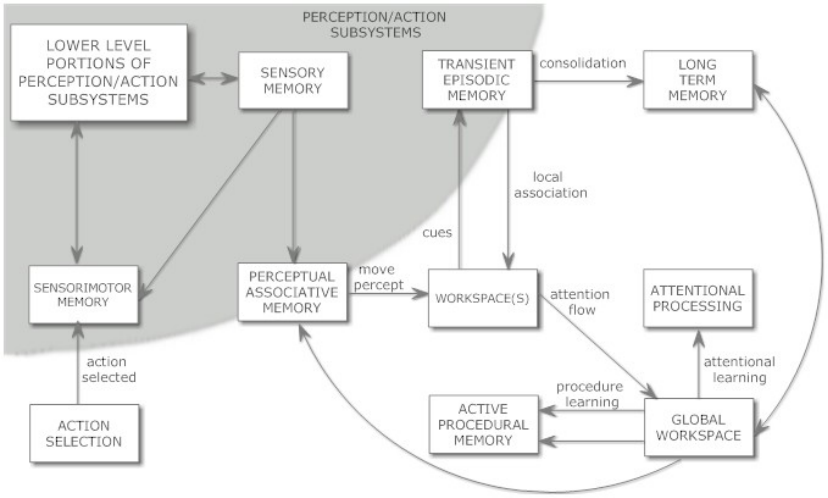


Fig. 5.2 Architecture of Working Memory and Reactive Processing, closely modeled on the LIDA architecture

term memory, which is related to but different from the one given in the integrative diagram – it views the multiple memory types distinguished in the integrative diagram as emergent from a common memory substrate. Psi comes with its own theory of perception and action, which seems broadly consistent with the deep learning approach incorporated in the integrative diagram. Psi's handling of working memory lacks the detailed, explicit workflow of LIDA, though it seems broadly conceptually consistent with LIDA.

In Figure 5.3, the box labeled "Other parts of working memory" is labeled "Protocol and situation memory" in the original Psi diagram. The Perception, Action Execution and Action Selection boxes have fairly similar semantics to the similarly labeled boxes in the LIDA-like Figure 5.2, so that these diagrams may be viewed as overlapping. The LIDA model doesn't explain action selection and planning in as much detail as Psi, so the Psi-like Figure 5.3 could be viewed as an elaboration of the action-selection portion of the LIDA-like Figure 5.2. In Psi, reinforcement is considered as part of the learning process involved in action selection and planning; in Figure 5.3 an explicit "reinforcement box" has been added to the original Psi diagram, to emphasize this.

Figure 5.4, modeling long-term memory and deliberative processing, is derived from our own prior work studying the "cognitive synergy" between different cognitive processes associated with different types of memory. The division into types of memory is fairly standard. Declarative, procedural, episodic and sensorimotor memory are routinely distinguished; we like to distinguish attentional memory and intentional (goal) memory as well, and view these as the interface between long-term memory and the mind's global

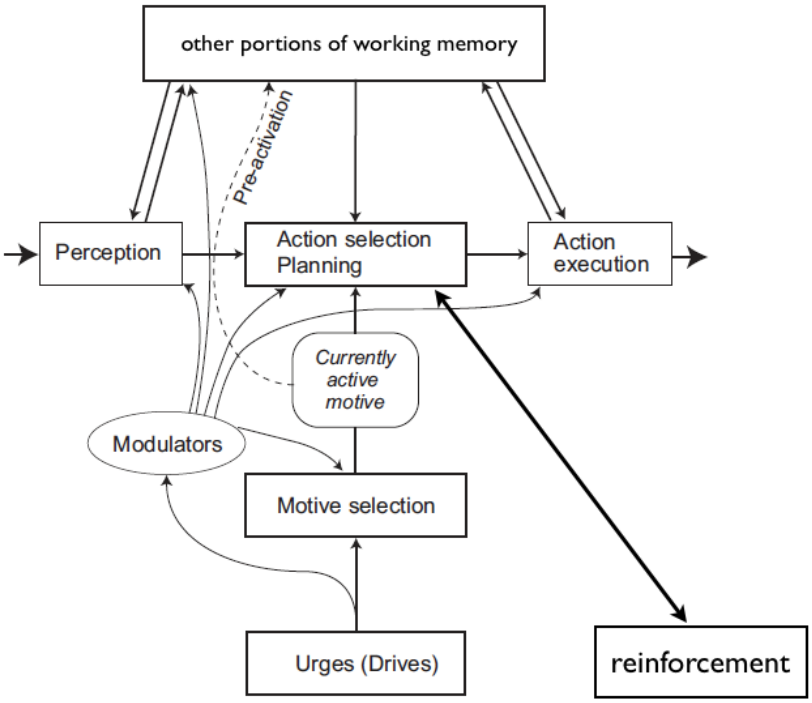


Fig. 5.3 Architecture of Motivated Action

control systems. One focus of our AGI design work has been on designing learning algorithms, corresponding to these various types of memory, that interact with each other in a synergetic way [Goe09b], helping each other to overcome their intrinsic combinatorial explosions. There is significant evidence that these various types of long-term memory are differently implemented in the brain, but the degree of structure and dynamical commonality underlying these different implementations remains unclear.

Each of these long-term memory types has its analogue in working memory as well. In some cognitive models, the working memory and long-term memory versions of a memory type and corresponding cognitive processes, are basically the same thing. CogPrime is mostly like this –it implements working memory as a subset of long-term memory consisting of items with particularly high importance values. The distinctive nature of working memory is enforced via using slightly different dynamical equations to update the importance values of items with importance above a certain threshold. On the other hand, many cognitive models treat working and long term memory as more distinct than this, and there is evidence for significant functional and anatomical distinctness in the brain in some cases. So for the purpose of the

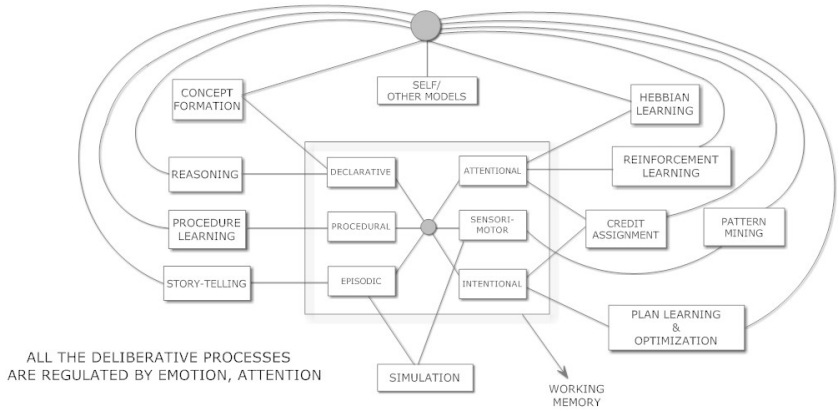


Fig. 5.4 Architecture of Long-Term Memory and Deliberative and Metacognitive Thinking

integrative diagram, it seemed best to leave working and long-term memory subcomponents as parallel but distinguished.

Figure 5.4 also encompasses metacognition, under the hypothesis that in human beings and human-like minds, metacognitive thinking is carried out using basically the same processes as plain ordinary deliberative thinking, perhaps with various tweaks optimizing them for thinking about thinking. If it turns out that humans have, say, a special kind of reasoning faculty exclusively for metacognition, then the diagram would need to be modified. Modeling of self and others is understood to occur via a combination of metacognition and deliberative thinking, as well as via implicit adaptation based on reactive processing.

Figure 5.5 models perception, according to the basic ideas of deep learning theory. Vision and audition are modeled as deep learning hierarchies, with bottom-up and top-down dynamics. The lower layers in each hierarchy refer to more localized patterns recognized in, and abstracted from, sensory data. Output from these hierarchies to the rest of the mind is not just through the top layers, but via some sort of sampling from various layers, with a bias toward the top layers. The different hierarchies cross-connect, and are hence to an extent dynamically coupled together. It is also recognized that there are some sensory modalities that aren't strongly hierarchical, e.g touch and smell (the latter being better modeled as something like an asymmetric Hopfield net, prone to frequent chaotic dynamics [LLW+05]) – these may also cross-connect with each other and with the more hierarchical perceptual subnetworks. Of course the suggested architecture could include any number of sensory modalities; the diagram is restricted to four just for simplicity.

The self-organized patterns in the upper layers of perceptual hierarchies may become quite complex and may develop advanced cognitive capabilities like episodic memory, reasoning, language learning, etc. A pure deep learn-

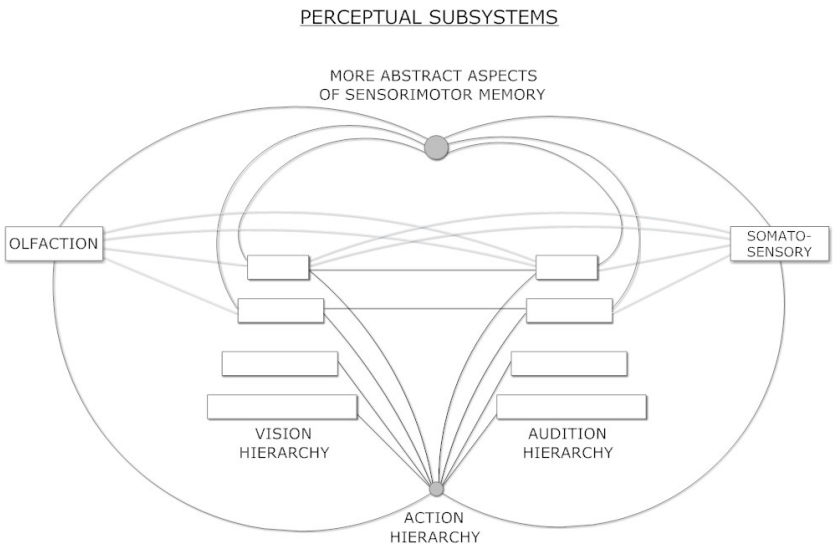


Fig. 5.5 Architecture for Multimodal Perception

ing approach to intelligence argues that all the aspects of intelligence emerge from this kind of dynamics (among perceptual, action and reinforcement hierarchies). Our own view is that the heterogeneity of human brain architecture argues against this perspective, and that deep learning systems are probably better as models of perception and action than of general cognition. However, the integrative diagram is not committed to our perspective on this – a deep-learning theorist could accept the integrative diagram, but argue that all the other portions besides the perceptual, action and reinforcement hierarchies should be viewed as descriptions of phenomena that emerge in these hierarchies due to their interaction.

Figure 5.6 shows an action subsystem and a reinforcement subsystem, parallel to the perception subsystem. Two action hierarchies, one for an arm and one for a leg, are shown for concreteness, but of course the architecture is intended to be extended more broadly. In the hierarchy corresponding to an arm, for example, the lowest level would contain control patterns corresponding to individual joints, the next level up to groupings of joints (like fingers), the next level up to larger parts of the arm (hand, elbow). The different hierarchies corresponding to different body parts cross-link, enabling coordination among body parts; and they also connect at multiple levels to perception hierarchies, enabling sensorimotor coordination. Finally there is a module for motor planning, which links tightly with all the motor hierarchies, and also overlaps with the more cognitive, inferential planning activities of the mind, in a manner that is modeled different ways by different theorists. Albus [AM01] has elaborated this kind of hierarchy quite elaborately.

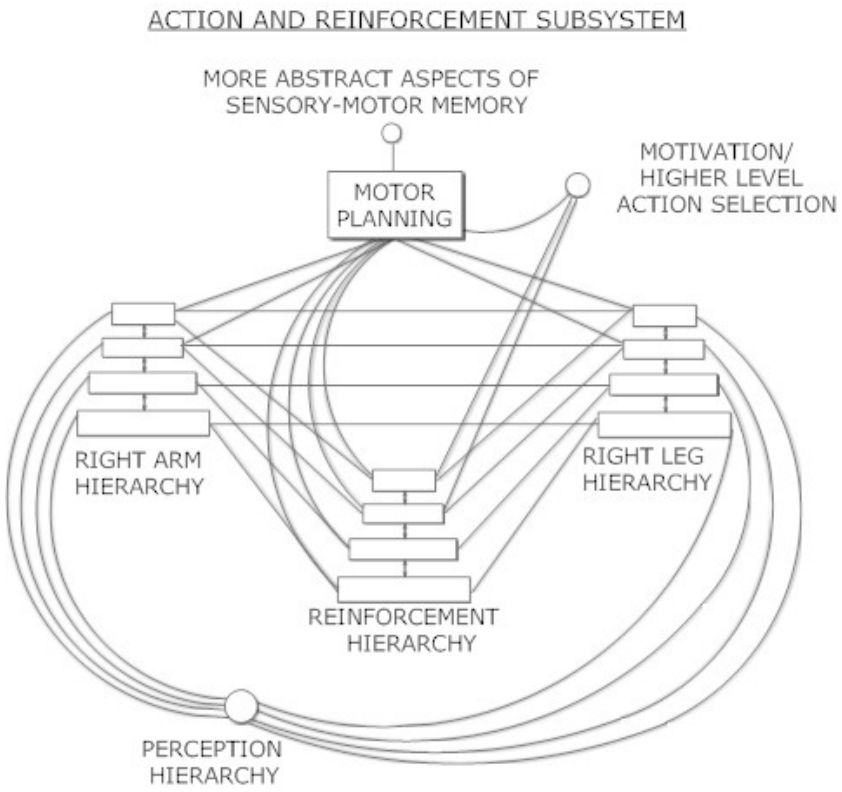


Fig. 5.6 Architecture for Action and Reinforcement

The reward hierarchy in Figure 5.6 provides reinforcement to actions at various levels on the hierarchy, and includes dynamics for propagating information about reinforcement up and down the hierarchy.

Figure 5.7 deals with language, treating it as a special case of coupled perception and action. The traditional architecture of a computational language comprehension system is a pipeline [JM09] [Goe10c], which is equivalent to a hierarchy with the lowest-level linguistic features (e.g. sounds, words) at the bottom, and the highest level features (semantic abstractions) at the top, and syntactic features in the middle. Feedback connections enable semantic and cognitive modulation of lower-level linguistic processing. Similarly, language generation is commonly modeled hierarchically, with the top levels being the ideas needing verbalization, and the bottom level corresponding to the actual sentence produced. In generation the primary flow is top-down, with bottom-up flow providing modulation of abstract concepts by linguistic surface forms.

So, that's it – an integrative architecture diagram for human-like general intelligence, split among 7 different pictures, formed by judiciously merging

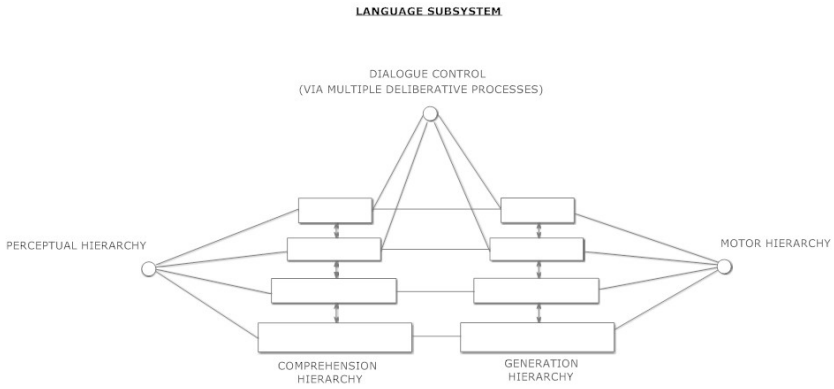


Fig. 5.7 Architecture for Language Processing

together architecture diagrams produced via a number of cognitive theorists with different, overlapping foci and research paradigms.

Is anything critical left out of the diagram? A quick perusal of the table of contents of cognitive psychology textbooks suggests to me that if anything major is left out, it's also unknown to current cognitive psychology. However, one could certainly make an argument for explicit inclusion of certain other aspects of intelligence, that in the integrative diagram are left as implicit emergent phenomena. For instance, creativity is obviously very important to intelligence, but, there is no "creativity" box in any of these diagrams – because in our view, and the view of the cognitive theorists whose work we've directly drawn on here, creativity is best viewed as a process emergent from other processes that are explicitly included in the diagrams.

5.4 Interpretation and Application of the Integrative Diagram

A tongue-partly-in-cheek definition of a biological pathway is "a subnetwork of a biological network, that fits on a single journal page." Cognitive architecture diagrams have a similar property – they are crude abstractions of complex structures and dynamics, sculpted in accordance with the size of the printed page, and the tolerance of the human eye for absorbing diagrams, and the tolerance of the human author for making diagrams.

However, sometimes constraints – even arbitrary ones – are useful for guiding creative efforts, due to the fact that they force choices. Creating an architecture for human-like general intelligence that fits in a few (okay, 7) fairly compact diagrams, requires one to make many choices about what features and relationships are most essential. In constructing the integrative diagram,

we have sought to make these choices, not purely according to our own tastes in cognitive theory or AGI system design, but according to a sort of blend of the taste and judgment of a number of scientists whose views we respect, and who seem to have fairly compatible, complementary perspectives.

What is the use of a cognitive architecture diagram like this? It can help to give newcomers to the field a basic idea about what is known and suspected about the nature of human-like general intelligence. Also, it could potentially be used as a tool for cross-correlating different AGI architectures. If everyone who authored an AGI architecture would explain how their architecture accounts for each of the structures and processes identified in the integrative diagram, this would give a means of relating the various AGI designs to each other.

The integrative diagram could also be used to help connect AGI and cognitive psychology to neuroscience in a more systematic way. In the case of LIDA, a fairly careful correspondence has been drawn up between the LIDA diagram nodes and links and various neural structures and processes [FB08]. Similar knowledge exists for the rest of the integrative diagram, though not organized in such a systematic fashion. A systematic curation of links between the nodes and links in the integrative diagram and current neuroscience knowledge, would constitute an interesting first approximation of the holistic cognitive behavior of the human brain.

Finally (and harking forward to later chapters), the big omission in the integrative diagram is *dynamics*. Structure alone will only get you so far, and you could build an AGI system with reasonable-looking things in each of the integrative diagram's boxes, interrelating according to the given arrows, and yet still fail to make a viable AGI system. Given the limitations the real world places on computing resources, it's not enough to have adequate representations and algorithms in all the boxes, communicating together properly and capable doing the right things given sufficient resources. Rather, one needs to have all the boxes filled in properly with structures and processes that, when they act together using feasible computing resources, will yield appropriately intelligent behaviors via their cooperative activity. And this has to do with the complex interactive dynamics of all the processes in all the different boxes – which is something the integrative diagram doesn't touch at all. This brings us again to the network of ideas we've discussed under the name of "cognitive synergy," to be discussed later on.

It might be possible to make something similar to the integrative diagram on the level of dynamics rather than structures, complementing the structural integrative diagram given here; but this would seem significantly more challenging, because we lack a standard set of tools for depicting system dynamics. Most cognitive theorists and AGI architects describe their structural ideas using boxes-and-lines diagrams of some sort, but there is no standard method for depicting complex system dynamics. So to make a dynamical analogue to the integrative diagram, via a similar integrative methodology, one would first need to create appropriate diagrammatic formalizations of the

dynamics of the various cognitive theories being integrated – a fascinating but onerous task.

When we first set out to make an integrated cognitive architecture diagram, via combining the complementary insights of various cognitive science and AGI theorists, we weren't sure how well it would work. But now we feel the experiment was generally a success – the resultant integrated architecture seems sensible and coherent, and reasonably complete. It doesn't come close to telling you everything you need to know to understand or implement a human-like mind – but it tells you the various processes and structures you need to deal with, and which of their interrelations are most critical. And, perhaps just as importantly, it gives a concrete way of understanding the insights of a specific but fairly diverse set of cognitive science and AGI theorists as complementary rather than contradictory. In a CogPrime context, it provides a way of tying in the specific structures and dynamics involved in CogPrime , with a more generic portrayal of the structures and dynamics of human-like intelligence.

Chapter 6
A Brief Overview of CogPrime

6.1 Introduction

Just as there are many different approaches to human flight – airplanes, helicopters, balloons, spacecraft, and doubtless many methods no person has thought of yet – similarly, there are likely many different approaches to advanced artificial general intelligence. All the different approaches to flight exploit the same core principles of aerodynamics in different ways; and similarly, the various different approaches to AGI will exploit the same core principles of general intelligence in different ways.

In the chapters leading up to this one, we have taken a fairly broad view of the project of engineering AGI. We have presented a conception and formal model of intelligence, and described environments, teaching methodologies and cognitive and developmental pathways that we believe are collectively appropriate for the creation of AGI at the human level and ultimately beyond, and with a roughly human-like bias to its intelligence. These ideas stand alone and may be compatible with a variety of approaches to engineering AGI systems. However, they also set the stage for the presentation of CogPrime , the particular AGI design on which we are currently working.

The thorough presentation of the CogPrime design is the job of Part 2 of this book – where, not only are the algorithms and structures involved in CogPrime reviewed in more detailed, but their relationship to the theoretical ideas underlying CogPrime is pursued more deeply. The job of this chapter is a smaller one: to give a high-level overview of some key aspects the CogPrime architecture at a mostly nontechnical level, so as to enable you to approach Part 2 with a little more idea of what to expect. The remainder of Part 1, following this chapter, will present various theoretical notions enabling the particulars, intent and consequences of the CogPrime design to be more thoroughly understood.

6.2 High-Level Architecture of CogPrime

Figures 6.1, 6.2 , 6.4 and 6.5 depict the high-level architecture of CogPrime , which involves the use of multiple cognitive processes associated with multiple types of memory to enable an intelligent agent to execute the procedures that it believes have the best probability of working toward its goals in its current context. In a robot preschool context, for example, the top-level goals will be simple things such as pleasing the teacher, learning new information and skills, and protecting the robot's body. Figure 6.3 shows part of the architecture via which cognitive processes interact with each other, via commonly acting on the AtomSpace knowledge repository.

Comparing these diagrams to the integrative human cognitive architecture diagram given in Chapter 5, one sees the main difference is that the CogPrime diagrams commit to specific structures (e.g. knowledge representations) and processes, whereas the generic integrative architecture diagram refers merely to types of structures and processes. For instance, the integrative diagram refers generally to declarative knowledge and learning, whereas the CogPrime diagram refers to PLN, as a specific system for reasoning and learning about declarative knowledge. Table 6.1 articulates the key connections between the components of the CogPrime diagram and those of the integrative diagram, thus indicating the general cognitive functions instantiated by each of the CogPrime components.

6.3 Current and Prior Applications of OpenCog

Before digging deeper into the theory, and elaborating some of the dynamics underlying the above diagrams, we pause to briefly discuss some of the practicalities of work done with the OpenCog system currently implementing parts of the CogPrime architecture.

OpenCog, the open-source software framework underlying the "OpenCogPrime " (currently partial) implementation of the CogPrime architecture, has been used for commercial applications in the area of natural language processing and data mining; for instance, see [**?**] where OpenCogPrimes PLN reasoning and RelEx language processing are combined to do automated biological hypothesis generation based on information gathered from PubMed abstracts. Most relevantly to the present work, it has also been used to control virtual agents in virtual worlds [GEA08].

Prototype work done during 2007-2008 involved using an OpenCog variant called the OpenPetBrain to control virtual dogs in a virtual world (see Figure 6.6 for a screenshot of an OpenPetBrain-controlled virtual dog). While these OpenCog virtual dogs did not display intelligence closely comparable to that of real dogs (or human children), they did demonstrate a variety of interesting and relevant functionalities including

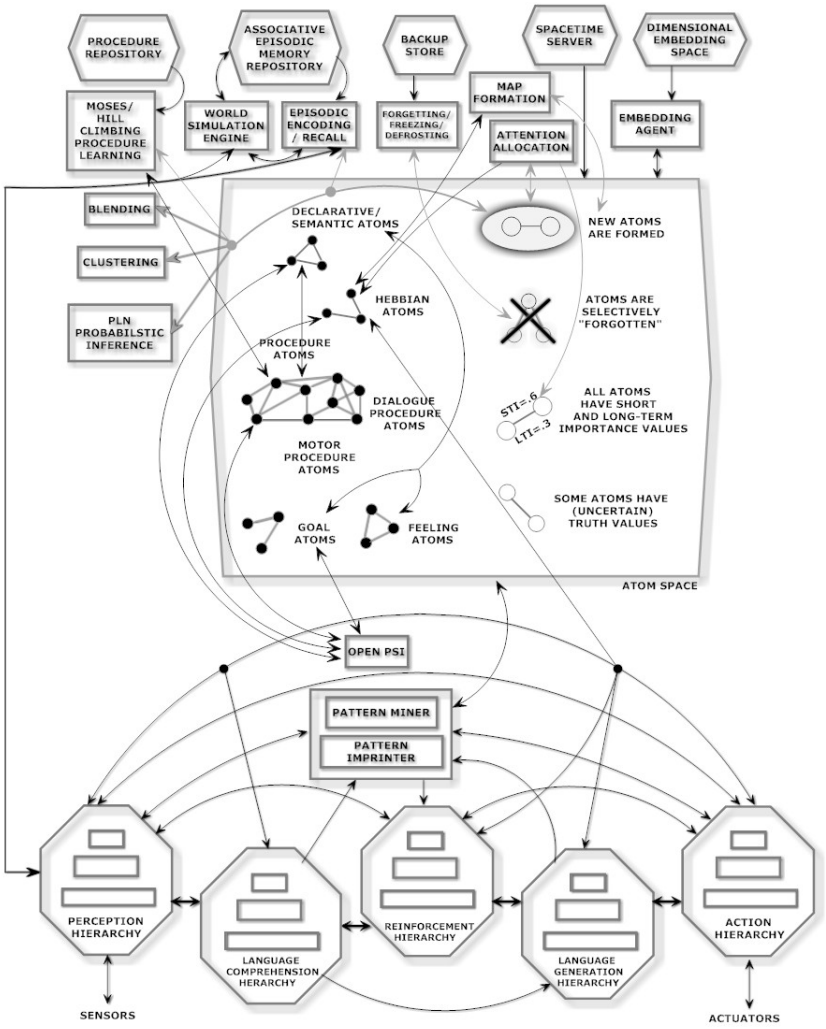


Fig. 6.1 High-Level Architecture of CogPrime . This is a conceptual depiction, not a detailed flowchart (which would be too complex for a single image). Figures 6.2 , 6.4 and 6.5 highlight specific aspects of this diagram.

- learning new behaviors based on imitation and reinforcement
- responding to natural language commands and questions, with appropriate actions and natural language replies
- spontaneous exploration of their world, remembering their experiences and using them to bias future learning and linguistic interaction

One current OpenCog initiative involves extending the virtual dog work via using OpenCog to control virtual agents in a game world inspired by

the game Minecraft. These agents are initially specifically concerned with achieving goals in a game world via constructing structures with blocks and carrying out simple English communications. Representative example tasks would be:

- Learning to build steps or ladders to get desired objects that are high up
- Learning to build a shelter to protect itself from aggressors
- Learning to build structures resembling structures that itÕs shown (even if the available materials are a bit different)
- Learning how to build bridges to cross chasms

Of course, the AI significance of learning tasks like this all depends on what kind of feedback the system is given, and how complex its environment is. It would be relatively simple to make an AI system do things like this in a trivial and highly specialized way, but that is not the intent of the project Ð the goal is to have the system learn to carry out tasks like this using general learning mechanisms and a general cognitive architecture, based on embodied experience and only scant feedback from human teachers. If successful, this will provide an outstanding platform for ongoing AGI development, as well as a visually appealing and immediately meaningful demo for OpenCog.

Specific, particularly simple tasks that are the focus of this project teamÕs current work at time of writing include:

- Watch another character build steps to reach a high-up object
- Figure out via imitation of this that, in a different context, building steps to reach a high up object may be a good idea
- Also figure out that, if it wants a certain high-up object but there are no materials for building steps available, finding some other way to get elevated will be a good idea that may help it get the object

6.3.1 Transitioning from Virtual Agents to a Physical Robot

Preliminary experiments have also been conducted using OpenCog to control a Nao robot as well as a virtual dog [GdG08]. This involves hybridizing OpenCog with a separate (but interlinked) subsystem handling low-level perception and action. In the experiments done so far, this has been accomplished in an extremely simplistic way. How to do this right is a topic treated in detail in Chapter 26 of Part 2.

We suspect that reasonable level of capability will be achievable by simply interposing DeSTIN (or some other system in its place) as a perception/action "black box" between OpenCog and a robot. Some preliminary experiments in this direction have already been carried out, connecting the OpenPetBrain

to a Nao robot using simpler, less capable software than DeSTIN in the intermediary role (off-the-shelf speech-to-text, text-to-speech and visual object recognition software).

However, we also suspect that to achieve robustly intelligent robotics we must go beyond this approach, and connect robot perception and actuation software with OpenCogPrime in a "white box" manner that allows intimate dynamic feedback between perceptual, motoric, cognitive and linguistic functions. We will achieve this via the creation and real-time utilization of links between the nodes in CogPrime 's and DeSTIN's internal networks (a topic to be explored in more depth later in this proposal).

6.4 Memory Types and Associated Cognitive Processes in CogPrime

Now we return to the basic description of the CogPrime approach, turning to aspects of the relationship between structure and dynamics. Architecture diagrams are all very well, but, ultimately it is dynamics that makes an architecture come alive. Intelligence is all about learning, which is by definition about change, about dynamical response to the environment and internal self-organizing dynamics.

CogPrime relies on multiple memory types and, as discussed above, is founded on the premise that the right course in architecting a pragmatic, roughly human-like AGI system is to handle different types of memory differently in terms of both structure and dynamics.

CogPrime 's memory types are the declarative, procedural, sensory, and episodic memory types that are widely discussed in cognitive neuroscience [TC05], plus attentional memory for allocating system resources generically, and intentional memory for allocating system resources in a goal-directed way. Table 6.2 overviews these memory types, giving key references and indicating the corresponding cognitive processes, and also indicating which of the generic patternist cognitive dynamics each cognitive process corresponds to (pattern creation, association, etc.). Figure 6.7 illustrates the relationships between several of the key memory types in the context of a simple situation involving an OpenCogPrime -controlled agent in a virtual world.

In terms of patternist cognitive theory, the multiple types of memory in CogPrime

should be considered as specialized ways of storing particular types of pattern, optimized for spacetime efficiency. The cognitive processes associated with a certain type of memory deal with creating and recognizing patterns of the type for which the memory is specialized. While in principle all the different sorts of pattern could be handled in a unified memory and processing architecture, the sort of specialization used in CogPrime is necessary in order to achieve acceptable efficient general intelligence using currently avail-

able computational resources. And as we have argued in detail in Chapter 7, efficiency is not a side-issue but rather the essence of real-world AGI (since as Hutter has shown, if one casts efficiency aside, arbitrary levels of general intelligence can be achieved via a trivially simple program).

The essence of the CogPrime design lies in the way the structures and processes associated with each type of memory are designed to work together in a closely coupled way, yielding cooperative intelligence going beyond what could be achieved by an architecture merely containing the same structures and processes in separate "black boxes."

The inter-cognitive-process interactions in OpenCog are designed so that

- conversion between different types of memory is possible, though sometimes computationally costly (e.g. an item of declarative knowledge may with some effort be interpreted procedurally or episodically, etc.)
- when a learning process concerned centrally with one type of memory encounters a situation where it learns very slowly, it can often resolve the issue by converting some of the relevant knowledge into a different type of memory: i.e. **cognitive synergy**

6.4.1 Cognitive Synergy in PLN

To put a little meat on the bones of the "cognitive synergy" idea, discussed repeatedly in prior chapters and more extensively in latter chapters, we now elaborate a little on the role it plays in the interaction between procedural and declarative learning.

While MOSES handles much of CogPrime 's procedural learning, and CogPrime 's internal simulation engine handles most episodic knowledge, CogPrime 's primary tool for handling declarative knowledge is an uncertain inference framework called Probabilistic Logic Networks (PLN). The complexities of PLN are the topic of a lengthy technical monograph [GMIH08], and are summarized in Chapter 34; here we will eschew most details and focus mainly on pointing out how PLN seeks to achieve efficient inference control via integration with other cognitive processes.

As a logic, PLN is broadly integrative: it combines certain term logic rules with more standard predicate logic rules, and utilizes both fuzzy truth values and a variant of imprecise probabilities called *indefinite probabilities*. PLN mathematics tells how these uncertain truth values propagate through its logic rules, so that uncertain premises give rise to conclusions with reasonably accurately estimated uncertainty values. This careful management of uncertainty is critical for the application of logical inference in the robotics context, where most knowledge is abstracted from experience and is hence highly uncertain.

PLN can be used in either forward or backward chaining mode; and in the language introduced above, it can be used for either analysis or synthesis.

As an example, we will consider backward chaining analysis, exemplified by the problem of a robot preschool-student trying to determine whether a new playmate "Bob" is likely to be a regular visitor to is preschool or not (evaluating the truth value of the implication $Bob \rightarrow regular\_visitor$). The basic backward chaining process for PLN analysis looks like:

1. Given an implication $L \equiv A \rightarrow B$ whose truth value must be estimated (for instance $L \equiv Concept \wedge Procedure \rightarrow Goal$ as discussed above), create a list $(A_1, ..., A_n)$ of *(inference rule, stored knowledge)* pairs that might be used to produce L
2. Using analogical reasoning to prior inferences, assign each A_i a probability of success
   - If some of the A_i are estimated to have reasonable probability of success at generating reasonably confident estimates of L's truth value, then invoke Step 1 with A_i in place of L (at this point the inference process becomes recursive)
   - If none of the A_i looks sufficiently likely to succeed, then inference has "gotten stuck" and another cognitive process should be invoked, e.g.
     - **Concept creation** may be used to infer new concepts related to A and B, and then Step 1 may be revisited, in the hope of finding a new, more promising A_i involving one of the new concepts
     - **MOSES** may be invoked with one of several special goals, e.g. the goal of finding a procedure P so that $P(X)$ predicts whether $X \rightarrow B$. If MOSES finds such a procedure P then this can be converted to declarative knowledge understandable by PLN and Step 1 may be revisited....
     - **Simulations** may be run in CogPrime 's internal simulation engine, so as to observe the truth value of $A \rightarrow B$ in the simulations; and then Step 1 may be revisited....

The combinatorial explosion of inference control is combatted by the capability to defer to other cognitive processes when the inference control procedure is unable to make a sufficiently confident choice of which inference steps to take next. Note that just as MOSES may rely on PLN to model its evolving populations of procedures, PLN may rely on MOSES to create complex knowledge about the terms in its logical implications. This is just one example of the multiple ways in which the different cognitive processes in CogPrime interact synergetically; a more thorough treatment of these interactions is given in [Goe09a].

In the "new playmate" example, the interesting case is where the robot initially seems not to know enough about Bob to make a solid inferential judgment (so that none of the A_i seem particularly promising). For instance, it might carry out a number of possible inferences and not come to any reasonably confident conclusion, so that the reason none of the A_i seem promising

is that all the decent-looking ones have been tried already. So it might then recourse to MOSES, simulation or concept creation.

For instance, the PLN controller could make a list of everyone who has been a regular visitor, and everyone who has not been, and pose MOSES the task of figuring out a procedure for distinguishing these two categories. This procedure could then used directly to make the needed assessment, or else be translated into logical rules to be used within PLN inference. For example, perhaps MOSES would discover that older males wearing ties tend not to become regular visitors. If the new playmate is an older male wearing a tie, this is directly applicable. But if the current playmate is wearing a tuxedo, then PLN may be helpful via reasoning that even though a tuxedo is not a tie, it's a similar form of fancy dress – so PLN may extend the MOSES-learned rule to the present case and infer that the new playmate is not likely to be a regular visitor.

6.5 Goal-Oriented Dynamics in CogPrime

CogPrime 's dynamics has both goal-oriented and "spontaneous" aspects; here for simplicity's sake we will focus on the goal-oriented ones. The basic goal-oriented dynamic of the CogPrime system, within which the various types of memory are utilized, is driven by implications known as "cognitive schematics", which take the form

$$Context \wedge Procedure \rightarrow Goal < p >$$

(summarized $C \wedge P \rightarrow G$). Semi-formally, this implication may be interpreted to mean: "If the context C appears to hold currently, then if I enact the procedure P, I can expect to achieve the goal G with certainty p." Cognitive synergy means that the learning processes corresponding to the different types of memory actively cooperate in figuring out what procedures will achieve the system's goals in the relevant contexts within its environment.

CogPrime 's cognitive schematic is significantly similar to production rules in classical architectures like SOAR and ACT-R (as reviewed in Chapter 4; however, there are significant differences which are important to CogPrime 's functionality. Unlike with classical production rules systems, uncertainty is core to CogPrime 's knowledge representation, and each CogPrime cognitive schematic is labeled with an uncertain truth value, which is critical to its utilization by CogPrime 's cognitive processes. Also, in CogPrime , cognitive schematics may be incomplete, missing one or two of the terms, which may then be filled in by various cognitive processes (generally in an uncertain way). A stronger similarity is to MicroPsi's triplets; the differences in this case are more low-level and technical and have already been mentioned in Chapter 4.

Finally, the biggest difference between CogPrime s cognitive schematics and production rules or other similar constructs, is that in CogPrime this level of knowledge representation is not the only important one. CLARION [SZ04], as reviewed above, is an example of a cognitive architecture that uses production rules for explicit knowledge representation and then uses a totally separate subsymbolic knowledge store for implicit knowledge. In CogPrime

both explicit and implicit knowledge are stored in the same graph of nodes and links, with

- explicit knowledge stored in probabilistic logic based nodes and links such as cognitive schematics (see Figure 6.8 for a depiction of some explicit linguistic knowledge.)
- implicit knowledge stored in patterns of activity among these same nodes and links, defined via the activity of the "importance" values (see Figure 6.9 for an illustrative example thereof) associated with nodes and links and propagated by the ECAN attention allocation process

The meaning of a cognitive schematic in CogPrime is hence not entirely encapsulated in its explicit logical form, but resides largely in the activity patterns that ECAN causes its activation or exploration to give rise to. And this fact is important because the synergetic interactions of system components are in large part modulated by ECAN activity. Without the real-time combination of explicit and implicit knowledge in the system's knowledge graph, the synergetic interaction of different cognitive processes would not work so smoothly, and the emergence of effective high-level hierarchical, heterarchical and self structures would be less likely.

6.6 Analysis and Synthesis Processes in CogPrime

We now return to CogPrime 's fundamental cognitive dynamics, using examples from the "virtual dog" application to motivate the discussion.

The cognitive schematic $Context \wedge Procedure \rightarrow Goal$ leads to a conceptualization of the internal action of an intelligent system as involving two key categories of learning:

- **Analysis**: Estimating the probability p of a posited $C \wedge P \rightarrow G$ relationship
- **Synthesis**: Filling in one or two of the variables in the cognitive schematic, given assumptions regarding the remaining variables, and directed by the goal of maximizing the probability of the cognitive schematic

More specifically, where synthesis is concerned,

- The MOSES probabilistic evolutionary program learning algorithm is applied to find P, given fixed C and G. Internal simulation is also used,

for the purpose of creating a simulation embodying C and seeing which P lead to the simulated achievement of G.

  - *Example: A virtual dog learns a procedure P to please its owner (the goal G) in the context C where there is a ball or stick present and the owner is saying "fetch".*

- PLN inference, acting on declarative knowledge, is used for choosing C, given fixed P and G (also incorporating sensory and episodic knowledge as appropriate). Simulation may also be used for this purpose.

  - *Example: A virtual dog wants to achieve the goal G of getting food, and it knows that the procedure P of begging has been successful at this before, so it seeks a context C where begging can be expected to get it food. Probably this will be a context involving a friendly person.*

- PLN-based goal refinement is used to create new subgoals G to sit on the right hand side of instances of the cognitive schematic.

  - *Example: Given that a virtual dog has a goal of finding food, it may learn a subgoal of following other dogs, due to observing that other dogs are often heading toward their food.*

- Concept formation heuristics are used for choosing G and for fueling goal refinement, but especially for choosing C (via providing new candidates for C). They are also used for choosing P, via a process called "predicate schematization" that turns logical predicates (declarative knowledge) into procedures.

  - *Example: At first a virtual dog may have a hard time predicting which other dogs are going to be mean to it. But it may eventually observe common features among a number of mean dogs, and thus form its own concept of "pit bull," without anyone ever teaching it this concept explicitly.*

Where analysis is concerned:

- PLN inference, acting on declarative knowledge, is used for estimating the probability of the implication in the cognitive schematic, given fixed C, P and G. Episodic knowledge is also used in this regard, via enabling estimation of the probability via simple similarity matching against past experience. Simulation is also used: multiple simulations may be run, and statistics may be captured therefrom.

  - *Example: To estimate the degree to which asking Bob for food (the procedure P is "asking for food", the context C is "being with Bob") will achieve the goal G of getting food, the virtual dog may study its memory to see what happened on previous occasions where it or*

other dogs asked Bob for food or other things, and then integrate the evidence from these occasions.

- Procedural knowledge, mapped into declarative knowledge and then acted on by PLN inference, can be useful for estimating the probability of the implication $C \wedge P \rightarrow G$, in cases where the probability of $C \wedge P_1 \rightarrow G$ is known for some P_1 related to P.
  - *Example: knowledge of the internal similarity between the procedure of asking for food and the procedure of asking for toys, allows the virtual dog to reason that if asking Bob for toys has been successful, maybe asking Bob for food will be successful too.*
- Inference, acting on declarative or sensory knowledge, can be useful for estimating the probability of the implication $C \wedge P \rightarrow G$, in cases where the probability of $C_1 \wedge P \rightarrow G$ is known for some C_1 related to C.
  - *Example: if Bob and Jim have a lot of features in common, and Bob often responds positively when asked for food, then maybe Jim will too.*
- Inference can be used similarly for estimating the probability of the implication $C \wedge P \rightarrow G$, in cases where the probability of $C \wedge P \rightarrow G_1$ is known for some G_1 related to G. Concept creation can be useful indirectly in calculating these probability estimates, via providing new concepts that can be used to make useful inference trails more compact and hence easier to construct.
  - *Example: The dog may reason that because Jack likes to play, and Jack and Jill are both children, maybe Jill likes to play too. It can carry out this reasoning only if its concept creation process has invented the concept of "child" via analysis of observed data.*

In these examples we have focused on cases where two terms in the cognitive schematic are fixed and the third must be filled in; but just as often, the situation is that only one of the terms is fixed. For instance, if we fix G, sometimes the best approach will be to collectively learn C and P. This requires either a procedure learning method that works interactively with a declarative-knowledge-focused concept learning or reasoning method; or a declarative learning method that works interactively with a procedure learning method. That is, it requires the sort of cognitive synergy built into the CogPrime design.

6.7 Conclusion

To thoroughly describe a comprehensive, integrative AGI architecture in a brief chapter would be an impossible task; all we have attempted here is a brief overview, to be elaborated on in the 500-odd pages of Part 2 of this book. We do not expect this brief summary to be enough to convince the skeptical reader that the approach described here has a reasonable odds of success at achieving its stated goals, or even of fulfilling the conceptual notions outlined in the preceding chapters. However, we hope to have given the reader at least a rough idea of *what sort of AGI design* we are advocating, and *why and in what sense we believe it can lead to advanced artificial general intelligence*. For more details on the structure, dynamics and underlying concepts of CogPrime , the reader is encouraged to proceed to Part 2– after completing Part 1, of course. Please be patient – building a thinking machine is a big topic, and we have a lot to say about it!

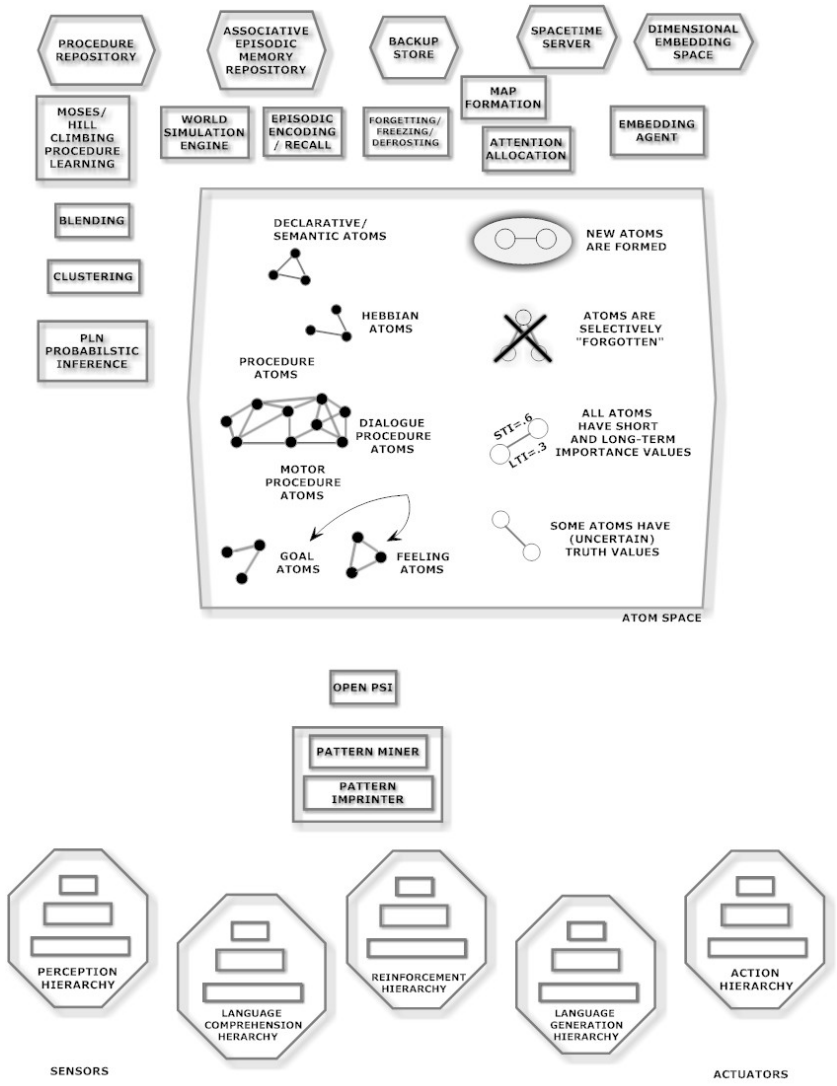


Fig. 6.2 Key Explicitly Implemented Processes of CogPrime . The large box at the center is the Atomspace, the system's central store of various forms of (long-term and working) memory, which contains a weighted labeled hypergraph whose nodes and links are "Atoms" of various sorts. The hexagonal boxes at the bottom denote various hierarchies devoted to recognition and generation of patterns: perception, action and linguistic. Intervening between these recognition/generation hierarchies and the Atomspace, we have a pattern mining/imprinting component (that recognizes patterns in the hierarchies and passes them to the Atomspace; and imprints patterns from the Atomspace on the hierarchies); and also OpenPsi, a special dynamical framework for choosing actions based on motivations. Above the Atomspace we have a host of cognitive processes, which act on the Atomspace, some continually and some only as context dictates, carrying out various sorts of learning and reasoning (pertinent to various sorts of memory) that help the system fulfill its goal sand motivations.

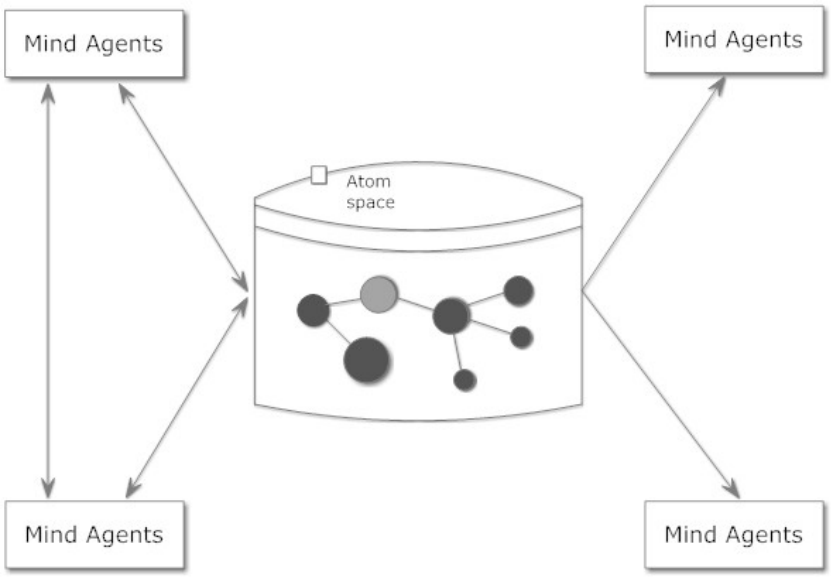


Fig. 6.3 MindAgents and AtomSpace in OpenCog. This is a conceptual depiction of one way cognitive processes may interact in OpenCog – they may be wrapped in MindAgent objects, which interact via cooperatively acting on the AtomSpace.

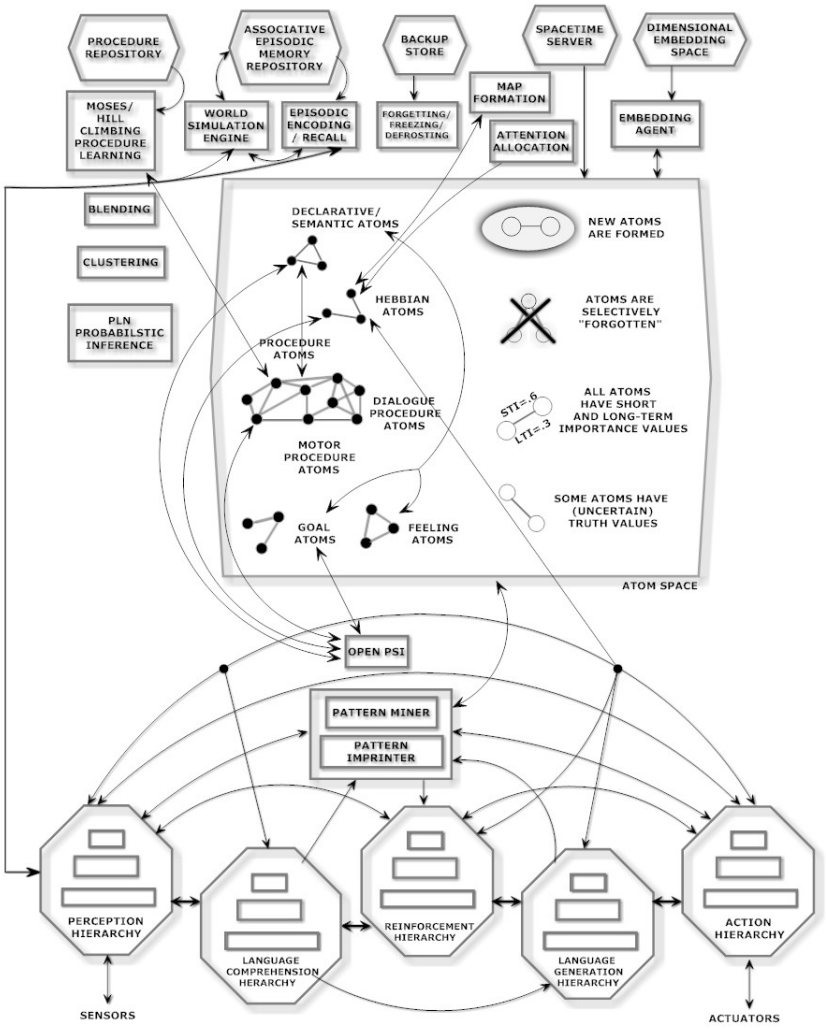


Fig. 6.4 Links Between Cognitive Processes and the Atomspace. The cognitive processes depicted all act on the Atomspace, in the sense that they operate by observing certain Atoms in the Atomspace and then modifying (or in rare cases deleting) them, and potentially adding new Atoms as well. Atoms represent all forms of knowledge, but some forms of knowledge are additionally represented by external data stores connected to the Atomspace, such as the Procedure Repository; these are also shown as linked to the Atomspace.

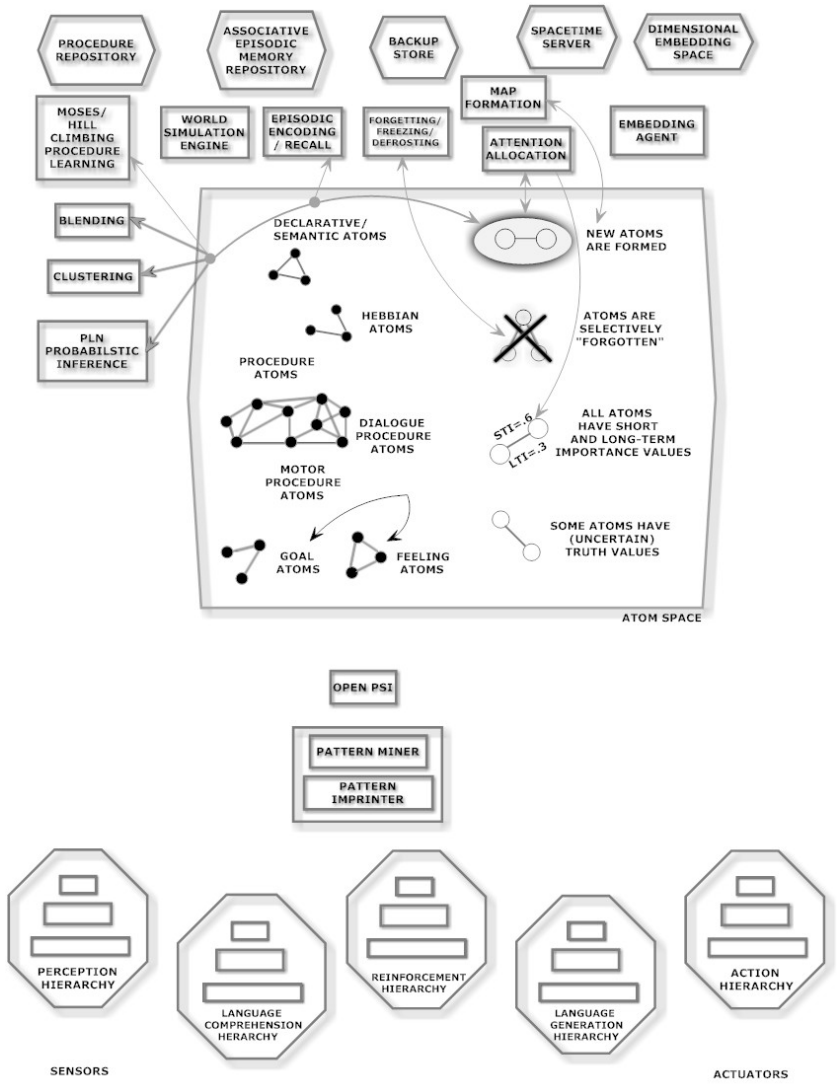


Fig. 6.5 Invocation of Atom Operations By Cognitive Processes. This diagram depicts some of the Atom modification, creation and deletion operations carried out by the abstract cognitive processes in the CogPrime architecture.

CogPrime Component	**Int. Diag. Sub-Diagram**	**Int. Diag. Component**
Procedure Repository	Long-Term Memory	Procedural
Procedure Repository	Working Memory	Active Procedural
Associative Episodic Memory	Long-Term Memory	Episodic
Associative Episodic Memory	Working Memory	Transient Episodic
Backup Store	Long-Term Memory	*no correlate: a function not necessarily possessed by the human mind*
Spacetime Server	Long-Term Memory	Declarative and Sensorimotor
Dimensional Embedding Space	*no clear correlate: a tool for helping multiple types of LTM*	
Dimensional Embedding Agent	*no clear correlate*	
Blending	Long-Term and Working Memory	Concept Formation
Clustering	Long-Term and Working Memory	Concept Formation
PLN Probabilistic Inference	Long-Term and Working Memory	Reasoning and Plan Learning/Optimization
MOSES / Hillclimbing	Long-Term and Working Memory	Procedure Learning
World Simulation	Long-Term and Working Memory	Simulation
Episodic Encoding / Recall	Long-Term g Memory	Story-telling
Episodic Encoding / Recall	Working Memory	Consolidation
Forgetting / Freezing / Defrosting	Long-Term and Working Memory	*no correlate: a function not necessarily possessed by the human mind*
Map Formation	Long-Term Memory	Concept Formation and Pattern Mining
Attention Allocation	Long-Term and Working Memory	Hebbian/Attentional Learning
Attention Allocation	High-Level Mind Architecture	Reinforcement
Attention Allocation	Working Memory	Perceptual Associative Memory and Local Association
AtomSpace	High-Level Mind Architecture	*no clear correlate: a general tool for representing memory including long-term and working, plus some of perception and action*
AtomSpace	Working Memory	Global Workspace (the high-STI portion of AtomSpace) & other Workspaces
Declarative Atoms	Long-Term and Working Memory	Declarative and Sensorimotor
Procedure Atoms	Long-Term and Working Memory	Procedural
Hebbian Atoms	Long-Term and Working Memory	Attentional
Goal Atoms	Long-Term and Working Memory	Intentional
Feeling Atoms	Long-Term and Working Memory	spanning Declarative, Intentional and Sensorimotor
OpenPsi	High-Level Mind Architecture	Motivation / Action Selection
OpenPsi	Working Memory	Action Selection
Pattern Miner	High-Level Mind Architecture	arrows between perception and working and long-term memory
Pattern Miner	Working Memory	arrows between sensory memory and perceptual associative and transient episodic memory
		arrows between action selection and

Fig. 6.6 Screenshot of OpenCog-controlled virtual dog

Memory Type	Specific Cognitive Processes	General Cognitive Functions
Declarative	Probabilistic Logic Networks (PLN) [GMIH08]; conceptual blending [FT02]	pattern creation
Procedural	MOSES (a novel probabilistic evolutionary program learning algorithm) [Loo06]	pattern creation
Episodic	internal simulation engine [GEA08]	association, pattern creation
Attentional	Economic Attention Networks (ECAN) [GPI+10]	association, credit assignment
Intentional	probabilistic goal hierarchy refined by PLN and ECAN, structured according to MicroPsi [Bac09]	credit assignment, pattern creation
Sensory	In CogBot, this will be supplied by the DeSTIN component	association, attention allocation, pattern creation, credit assignment

Table 6.2 Memory Types and Cognitive Processes in CogPrime . The third column indicates the general cognitive function that each specific cognitive process carries out, according to the patternist theory of cognition.

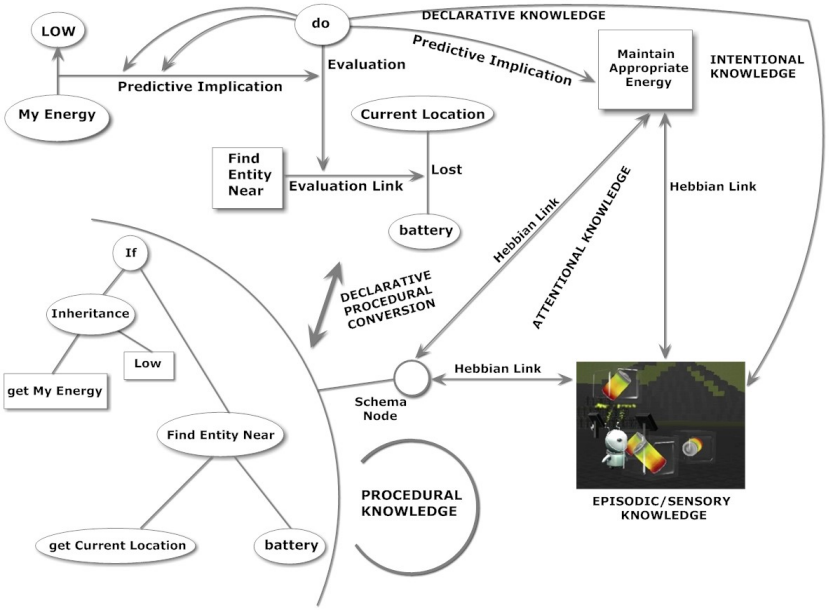


Fig. 6.7 Relationship Between Multiple Memory Types. The bottom left corner shows a program tree, constituting procedural knowledge. The upper left shows declarative nodes and links in the Atomspace. The upper right corner shows a relevant system goal. The lower right corner contains an image symbolizing relevant episodic and sensory knowledge. All the various types of knowledge link to each other and can be approximatively converted to each other.

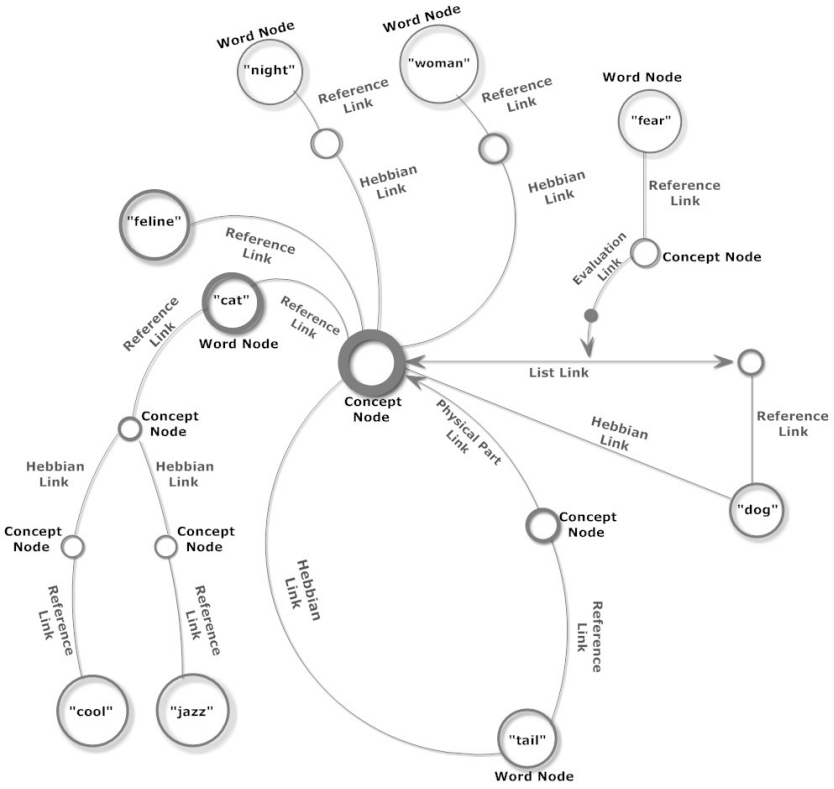


Fig. 6.8 Example of Explicit Knowledge in the Atomspace. One simple example of explicitly represented knowledge in the Atomspace is linguistic knowledge, such as words and the concepts directly linked to them. Not all of a CogPrime system's concepts correlate to words, but some do.

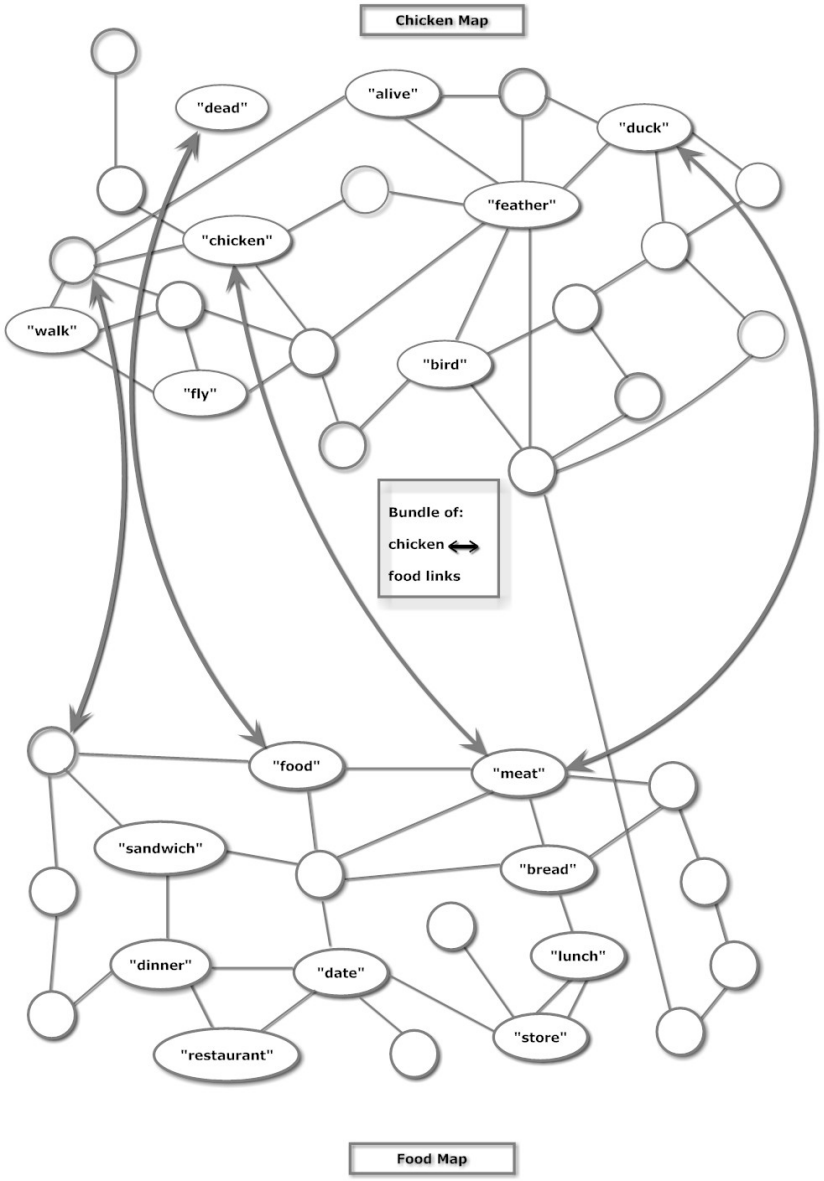


Fig. 6.9 Example of Implicit Knowledge in the Atomspace. A simple example of implicit knowledge in the Atomspace. The "chicken" and "food" concepts are represented by "maps" of ConceptNodes interconnected by HebbianLinks, where the latter tend to form between ConceptNodes that are often simultaneously important. The bundle of links between nodes in the chicken map and nodes in the food map, represents an "implicit, emergent link" between the two concept maps. This diagram also illustrates "glocal" knowledge representation, in that the chicken and food concepts are each represented by individual nodes, but also by distributed maps. The "chicken" ConceptNode, when important, will tend to make the rest of the map important – and vice versa. Part of the overall chicken concept possessed by the system is expressed by the explicit links coming out of the chicken ConceptNode, and part is represented only by the distributed chicken map as a whole.

Section II
Toward a General Theory of General Intelligence

Chapter 7
A Formal Model of Intelligent Agents

7.1 Introduction

The artificial intelligence field is full of sophisticated mathematical models and equations, but most of these are highly specialized in nature – e.g. formalizations of particular logic systems, analyzes of the dynamics of specific sorts of neural nets, etc. On the other hand, a number of highly general models of intelligent systems also exist, including Hutter's recent formalization of universal intelligence [Hut05] and a large body of work in the disciplines of systems science and cybernetics – but these have tended not to yield many specific lessons useful for engineering AGI systems, serving more as conceptual models in mathematical form.

It would be fantastic to have a mathematical theory bridging these extremes – a real "general theory of general intelligence," allowing the derivation and analysis of specific structures and processes playing a role in practical AGI systems, from broad mathematical models of general intelligence in various situations and under various constraints. However, the path to such a theory is not entirely clear at present; and, as valuable as such a theory would be, we don't believe such a thing to be *necessary* for creating advanced AGI. One possibility is that the development of such a theory will occur contemporaneously and synergetically with the advent of practical AGI technology.

Lacking a mature, pragmatically useful "general theory of general intelligence," however, we have still found it valuable to articulate certain theoretical ideas about the nature of general intelligence, with a level of rigor a bit greater than the wholly informal discussions of the previous chapters. The chapters in this section of the book articulate some ideas we have developed in pursuit of a general theory of general intelligence; ideas that, even in their current relatively undeveloped form, have been very helpful in guiding our concrete work on the CogPrime design.

This chapter presents a more formal version of the notion of intelligence as "achieving complex goals in complex environments," based on a formal

model of intelligent agents. These formalizations of agents and intelligence will be used in later chapters as a foundation for formalizing other concepts like inference and cognitive synergy. Chapters 8 and 9 pursue the notion of cognitive synergy a little more thoroughly than was done in previous chapters. Chapter 10 sketches a general theory of general intelligence using tools from category theory – not bringing it to the level where one can use it to derive specific AGI algorithms and structures; but still, presenting ideas that will be helpful in interpreting and explaining specific aspects of the CogPrime design in Part 2. Finally, Appendix B explores an additional theoretical direction, in which the mind of an intelligent system is viewed in terms of certain curved spaces – a novel way of thinking about the dynamics of general intelligence, which has been useful in guiding development of the ECAN component of CogPrime , and we expect will have more general value in future.

Despite the intermittent use of mathematical formalism, the ideas presented in this section are fairly speculative, and we do not propose them as constituting a well-demonstrated theory of general intelligence. Rather, we propose them as an interesting *way of thinking* about general intelligence, which appears to be consistent with available data, and which has proved inspirational to us in conceiving concrete structures and dynamics for AGI, as manifested for example in the CogPrime design. Understanding the way of thinking described in these chapters is valuable for understanding why the CogPrime design is the way it is, and for relating CogPrime to other practical and intellectual systems, and extending and improving CogPrime .

7.2 A Simple Formal Agents Model (SRAM)

We now present a formalization of the concept of "intelligent agents" – beginning with a formalization of "agents" in general.

Drawing on [Hut05, LH07], we consider a class of active agents which observe and explore their environment and also take actions in it, which may affect the environment. Formally, the agent sends information to the environment by sending symbols from some finite alphabet called the *action space* Σ; and the environment sends signals to the agent with symbols from an alphabet called the *perception space*, denoted $\mathcal{P}$. Agents can also experience rewards, which lie in the *reward space*, denoted $\mathcal{R}$, which for each agent is a subset of the rational unit interval.

The agent and environment are understood to take turns sending signals back and forth, yielding a history of actions, observations and rewards, which may be denoted

$$a_1 o_1 r_1 a_2 o_2 r_2 ...$$

or else

$$a_1 x_1 a_2 x_2 ...$$

if x is introduced as a single symbol to denote both an observation and a reward. The complete interaction history up to and including cycle t is denoted $ax_{1:t}$; and the history before cycle t is denoted $ax_{<t} = ax_{1:t-1}$.

The agent is represented as a function π which takes the current history as input, and produces an action as output. Agents need not be deterministic, an agent may for instance induce a probability distribution over the space of possible actions, conditioned on the current history. In this case we may characterize the agent by a probability distribution $\pi(a_t|ax_{<t})$. Similarly, the environment may be characterized by a probability distribution $\mu(x_k|ax_{<k}a_k)$. Taken together, the distributions π and μ define a probability measure over the space of interaction sequences.

Next, we extend this model in a few ways, intended to make it better reflect the realities of intelligent computational agents. The first modification is to allow agents to maintain memories (of finite size), via adding memory actions drawn from a set $\mathcal{M}$ into the history of actions, observations and rewards. The second modification is to introduce the notion of **goals**.

7.2.1 Goals

We define goals as mathematical functions (to be specified below) associated with symbols drawn from the alphabet $\mathcal{G}$; and we consider the environment as sending goal-symbols to the agent along with regular observation-symbols. (Note however that the presentation of a goal-symbol to an agent does not necessarily entail the explicit communication to the agent of the contents of the goal function. This must be provided by other, correlated observations.) We also introduce a conditional distribution $\gamma(g, \mu)$ that gives the weight of a goal g in the context of a particular environment μ.

In this extended framework, an interaction sequence looks like

$$a_1 o_1 g_1 r_1 a_2 o_2 g_2 r_2 ...$$

or else

$$a_1 y_1 a_2 y_2 ...$$

where g_i are symbols corresponding to goals, and y is introduced as a single symbol to denote the combination of an observation, a reward and a goal.

Each goal function maps each finite interaction sequence $I_{g,s,t} = ay_{s:t}$ with g_s to g_t corresponding to g, into a value $r_g(I_{g,s,t}) \in [0, 1]$ indicating the value or "raw reward" of achieving the goal during that interaction sequence. The

total reward r_t obtained by the agent is the sum of the raw rewards obtained at time t from all goals whose symbols occur in the agent's history before t.

This formalism of goal-seeking agents allows us to formalize the notion of intelligence as "achieving complex goals in complex environments" – a direction that is pursued in Section 7.3 below.

7.2.2 Memory Stores

As well as goals, we introduce into the model a long-term memory and a workspace. Regarding long-term memory we assume the agent's memory consists of multiple memory stores corresponding to various types of memory, e.g.: procedural (K_{Proc}), declarative (K_{Dec}), episodic (K_{Ep}), attentional (K_{Att}) and Intentional (K_{Int}). In Appendix **??** a category-theoretic model of these memory stores is introduced; but for the moment, we need only assume the existence of

- an injective mapping $\Theta_{Ep} : K_{Ep} \to \mathcal{H}$ where $\mathcal{H}$ is the space of fuzzy sets of subhistories (subhistories being "episodes" in this formalism)
- an injective mapping $\Theta_{Proc} : K_{Proc} \times \mathcal{M} \times \mathcal{W} \to \mathcal{A}$, where $\mathcal{M}$ is the set of memory states, $\mathcal{W}$ is the set of (observation, goal, reward) triples, and $\mathcal{A}$ is the set of actions (this maps each procedure object into a function that enacts actions in the environment or memory, based on the memory state and current world-state)
- an injective mapping $\Theta_{Dec} : K_{Dec} \to \mathcal{L}$, where $\mathcal{L}$ is the set of expressions in some formal language (which may for example be a logical language), which possesses words corresponding to the observations, goals, reward values and actions in our agent formalism
- an injective mapping $\Theta_{Int} : K_{Int} \to \mathcal{G}$, where $\mathcal{G}$ is the space of goals mentioned above
- an injective mapping $\Theta_{Att} : K_{Int} \cup K_{Ep} \cup K_{Proc} \cup K_{Ec} \to \mathcal{V}$, where $\mathcal{V}$ is the space of "attention values" (structures that gauge the importance of paying attention to an item of knowledge over various time-scales or in various contexts)

We also assume that the vocabulary of actions contains memory-actions corresponding to the operations of inserting the current observation, goal, reward or action into the episodic and/or declarative memory store. And, we assume that the activity of the agent, at each time-step, includes the enaction of one or more of the procedures in the procedural memory store. If several procedures are enacted at once, then the end result is still formally modeled as a single action $a = a_{[1]} * ... * a_{[k]}$ where $*$ is an operator on action-space that composes multiple actions into a single one.

Finally, we assume that, at each time-step, the agent may carry out an external action a_i on the environment, a memory action m_i on the (long-

term) memory, and an action b_i on its **internal workspace**. Among the actions that can be carried out on the workspace, are the ability to insert or delete observations, goals, actions or reward-values from the workspace. The workspace can be thought of as a sort of short-term memory or else in terms of Baars' "global workspace" concept mentioned above. The workspace provides a medium for interaction between the different memory types.

The workspace provides a mechanism by which declarative, episodic and procedural memory may interact with each other. For this mechanism to work, we must assume that there are actions corresponding to query operations that allow procedures to look into declarative and episodic memory. The nature of these query operations will vary among different agents, but we can assume that in general an agent has

- one or more procedures $Q_{Dec}(x)$ serving as *declarative queries*, meaning that when Q_{Dec} is enacted on some x that is an ordered set of items in the workspace, the result is that one or more items from declarative memory is entered into the workspace
- one or more procedures $Q_{Ep}(x)$ serving as *episodic queries*, meaning that when Q_{Ep} is enacted on some x that is an ordered set of items in the workspace, the result is that one or more items from episodic memory is entered into the workspace

One additional aspect of CogPrime 's knowledge representation that is important to PLN is the attachment of nonnegative weights n_i corresponding to elementary observations o_i. These weights denote the amount of evidence contained in the observation. For instance, in the context of a robotic agent, one could use these values to encode the assumption that an elementary visual observation has more evidential value than an elementary olfactory observation.

We now have a model of an agent with long-term memory comprising procedural, declarative and episodic aspects, an internal cognitive workspace, and the capability to use procedures to drive actions based on items in memory and the workspace, and to move items between long-term memory and the workspace.

7.2.2.1 Modeling CogPrime

Of course, this formal model may be realized differently in various real-world AGI systems. In CogPrime we have

- a weighted, labeled hypergraph structure called the AtomSpace used to store declarative knowledge (this is the representation used by PLN)
- a collection of programs in a LISP-like language called Combo, stored in a ProcedureRepository data structure, used to store procedural knowledge
- a collection of partial "movies" of the system's experience, played back using an internal simulation engine, used to store episodic knowledge

- AttentionValue objects, minimally containing ShortTermImportance (STI) and LongTermImportance (LTI) values used to store attentional knowledge
- Goal Atoms for intentional knowledge, stored in the same format as declarative knowledge but whose dynamics involve a special form of artificial currency that is used to govern action selection

The AtomSpace is the central repository and procedures and episodes are linked to Atoms in the AtomSpace which serve as their symbolic representatives. The "workspace" in CogPrime exists only virtually: each item in the AtomSpace has a "short term importance" (STI) level, and the workspace consists of those items in the AtomSpace with highest STI, and those procedures and episodes whose symbolic representatives in the AtomSpace have highest STI.

On the other hand, as we saw above, the LIDA architecture uses separate representations for procedural, declarative and episodic memory, but also has an explicit workspace component, where the most currently contextually relevant items from all different types of memory are gathered and used together in the course of actions. However, compared to CogPrime , it lacks comparably fine-grained methods for integrating the different types of memory.

Systematically mapping various existing cognitive architectures, or human brain structure, into this formal agents model would be a substantial though quite plausible exercise; but we will not undertake this here.

7.2.3 The Cognitive Schematic

Next we introduce an additional specialization into SRAM: the **cognitive schematic**, written informally as

$$Context \ \& \ Procedure \rightarrow Goal$$

and considered more formally as $holds(C)$ & $ex(P) \rightarrow h_i$ where h may be an externally specified goal g_i or an internally specified goal h derived as a (possibly uncertain) subgoal of one of more g_i; C is a piece of declarative or episodic knowledge and P is a procedure that the agent can internally execute to generate a series of actions. $ex(P)$ is the proposition that P is successfully executed. If C is episodic then $holds(C)$ may be interpreted as the current context (i.e. some finite slice of the agent's history) being similar to C; if C is declarative then $holds(C)$ may be interpreted as the truth value of C evaluated at the current context. Note that C may refer to some part of the world quite distant from the agent's current sensory observations; but it may still be formally evaluated based on the agent's history.

In the standard CogPrime notation as introduced formally in Chapter 20 (where indentation has function-argument syntax similar to that in Python, and relationship types are prepended to their relata without parentheses), for the case C is declarative this would be written as

```
PredictiveExtensionalImplication
    AND
        C
        Execution P
    G
```

and in the case C is episodic one replaces C in this formula with a predicate expressing C's similarity to the current context. The semantics of the PredictiveExtensionalInheritance relation will be discussed below. The Execution relation simply denotes the proposition that procedure P has been executed.

For the class of SRAM agents who (like CogPrime) use the cognitive schematic to govern many or all of their actions, a significant fragment of agent intelligence boils down to estimating the truth values of PredictiveExtensionalImplication relationships. Action selection procedures can be used, which choose procedures to enact based on which ones are judged most likely to achieve the current external goals g_i in the current context. Rather than enter into the particularities of action selection or other cognitive architecture issues, we will restrict ourselves to PLN inference, which in the context of the present agent model is a method for handling PredictiveImplication in the cognitive schematic.

Consider an agent in a virtual world, one of whose external goals is to please its owner. Suppose its owner has asked it to find a cat, and it can translate this into a subgoal "find cat." If the agent operates according to the cognitive schematic, it will search for P so that

```
PredictiveExtensionalImplication
    AND
        C
        Execution P
    Evaluation
        found
        cat
```

holds.

7.3 Toward a Formal Characterization of Real-World General Intelligence

Having defined what we mean by an agent acting in an environment, we now turn to the question of what it means for such an agent to be "intelligent."

As we have reviewed extensively in Chapter 2 above, "intelligence" is a commonsense, "folk psychology" concept, with all the imprecision and contextuality that this generally entails. One cannot expect any compact, elegant formalism to capture all of its meanings. Even in the psychology and AI research communities, divergent definitions abound; Legg and Hutter [LH07] lists and organizes 70+ definitions from the literature.

Practical study of natural intelligence in humans and other organisms, and practical design, creation and instruction of artificial intelligences, can proceed perfectly well without an agreed-upon formalization of the "intelligence" concept. Some researchers may conceive their own formalisms to guide their own work, others may feel no need for any such thing.

But nevertheless, it is of interest to seek formalizations of the concept of intelligence, which capture useful fragments of the commonsense notion of intelligence, and provide guidance for practical research in cognitive science and AI. A number of such formalizations have been given in recent decades, with varying degrees of mathematical rigor. Perhaps the most carefully-wrought formalization of intelligence so far is the theory of "universal intelligence" presented by Shane Legg and Marcus Hutter in [**?**], which draws on ideas from algorithmic information theory.

Universal intelligence captures a certain aspect of the "intelligence" concept very well, and has the advantage of connecting closely with ideas in learning theory, decision theory and computation theory. However, the kind of general intelligence it captures best, is a kind which is in a sense *more general* in scope than human-style general intelligence. Universal intelligence does capture the sense in which humans are more intelligent than worms, which are more intelligent than rocks; and the sense in which theoretical AGI systems like Hutter's AIXI or $AIXI^{tl}$ [Hut05] would be much more intelligent than humans. But it misses essential aspects of the intelligence concept as it is used in the context of intelligent natural systems like humans or real-world AI systems.

Our main goal in this section is to present variants of universal intelligence that better capture the notion of intelligence as it is typically understood in the context of real-world natural and artificial systems. The first variant we describe is *pragmatic general intelligence*, which is inspired by the intuitive notion of intelligence as "the ability to achieve complex goals in complex environments," given in [**?**]. After assuming a prior distribution over the space of possible environments, and one over the space of possible goals, one then defines the pragmatic general intelligence as the expected level of goal-achievement of a system relative to these distributions. Rather than

measuring truly broad mathematical general intelligence, pragmatic general intelligence measures intelligence in a way that's specifically biased toward certain environments and goals.

Another variant definition is then presented, the *efficient pragmatic general intelligence*, which takes into account the amount of computational resources utilized by the system in achieving its intelligence. Some argue that making efficient use of available resources is a defining characteristic of intelligence, see e.g. [Wan06].

A critical question left open is the characterization of the prior distributions corresponding to everyday human reality; we give a semi-formal sketch of some ideas on this Chapter 9 below, where we present the notion of a "communication prior," which assigns a probability weight to a situation S based on the ease with which one agent in a society can communicate S to another agent in that society, using multimodal communication (including verbalization, demonstration, dramatic and pictorial depiction, etc.).

Finally, we present a formal measure of the "generality" of an intelligence, which precisiates the informal distinction between "general AI" and "narrow AI."

7.3.1 Biased Universal Intelligence

To define universal intelligence, Legg and Hutter consider the class of environments that are *reward-summable*, meaning that the total amount of reward they return to any agent is bounded by 1. Where r_i denotes the reward experienced by the agent from the environment at time i, the *expected total reward* for the agent π from the environment μ is defined as

$$V_\mu^\pi \equiv E(\sum_1^\infty r_i) \leq 1$$

To extend their definition in the direction of greater realism, we first introduce a second-order probability distribution ν, which is a probability distribution over the space of environments μ. The distribution ν assigns each environment a probability. One such distribution ν is the Solomonoff-Levin universal distribution in which one sets $\nu = 2^{-K(\mu)}$; but this is not the only distribution ν of interest. In fact a great deal of real-world general intelligence consists of the adaptation of intelligent systems to particular distributions ν over environment-space, differing from the universal distribution.

We then define

Definition 4 *The* **biased universal intelligence** *of an agent π is its expected performance with respect to the distribution ν over the space of all computable reward-summable environments, E, that is,*

$$\Upsilon(\pi) \equiv \sum_{\mu \in E} \nu(\mu) V_\mu^\pi$$

Legg and Hutter's **universal intelligence** is obtained by setting ν equal to the universal distribution.

This framework is more flexible than it might seem. E.g. suppose one wants to incorporate agents that die. Then one may create a special action, say a_{666}, corresponding to the state of death, to create agents that

- in certain circumstances output action a_{666}
- have the property that if their previous action was a_{666}, then all of their subsequent actions must be a_{666}

and to define a reward structure so that actions a_{666} always bring zero reward. It then follows that death is generally a bad thing if one wants to maximize intelligence. Agents that die will not get rewarded after they're dead; and agents that live only 70 years, say, will be restricted from getting rewards involving long-term patterns and will hence have specific limits on their intelligence.

7.3.2 Connecting Legg and Hutter's Model of Intelligent Agents to the Real World

A notable aspect of the Legg and Hutter formalism is the separation of the reward mechanism from the cognitive mechanisms of the agent. While commonplace in the reinforcement learning literature, this seems psychologically unrealistic in the context of biological intelligences and many types of machine intelligences. Not all human intelligent activity is specifically reward-seeking in nature; and even when it is, humans often pursue complexly constructed rewards, that are defined in terms of their own cognitions rather than separately given. Suppose a certain human's goals are true love, or world peace, and the proving of interesting theorems – then these goals are defined by the human herself, and only she knows if she's achieved them. An externally-provided reward signal doesn't capture the nature of this kind of goal-seeking behavior, which characterizes much human goal-seeking activity (and will presumably characterize much of the goal-seeking activity of advanced engineered intelligences also) ... let alone human behavior that is spontaneous and unrelated to explicit goals, yet may still appear commonsensically intelligent.

One could seek to bypass this complaint about the reward mechanisms via a sort of "neo-Freudian" argument, via

- associating the reward signal, not with the "external environment" as typically conceived, but rather with a portion of the intelligent agent's brain that is separate from the cognitive component

- viewing complex goals like true love, world peace and proving interesting theorems as indirect ways of achieving the agent's "basic goals", created within the agent's memory via subgoaling mechanisms

but it seems to us that a general formalization of intelligence should not rely on such strong assumptions about agents' cognitive architectures. So below, after introducing the pragmatic and efficient pragmatic general intelligence measures, we will propose an alternate interpretation wherein the mechanism of external rewards is viewed as a theoretical test framework for assessing agent intelligence, rather than a hypothesis about intelligent agent architecture.

In this alternate interpretation, formal measures like the universal, pragmatic and efficient pragmatic general intelligence are viewed as *not* directly applicable to real-world intelligences, because they involve the behaviors of agents over a wide variety of goals and environments, whereas in real life the opportunities to observe agents are more limited. However, they are viewed as being *indirectly* applicable to real-world agents, in the sense that an external intelligence can observe an agent's real-world behavior and then *infer* its likely intelligence according to these measures.

In a sense, this interpretation makes our formalized measures of intelligence the opposite of real-world IQ tests. An IQ test is a quantified, formalized test which is designed to approximately predict the informal, qualitative achievement of humans in real life. On the other hand, the formal definitions of intelligence we present here are quantified, formalized tests that are designed to capture abstract notions of intelligence, but which can be approximately evaluated on a real-world intelligent system by observing what it does in real life.

7.3.3 Pragmatic General Intelligence

The above concept of biased universal intelligence is perfectly adequate for many purposes, but it is also interesting to explicitly introduce the notion of a *goal* into the calculation. This allows us to formally capture the notion presented in [**?**] of intelligence as "the ability to achieve complex goals in complex environments."

If the agent is acting in environment μ, and is provided with g_s corresponding to g at the start and the end of the time-interval $T = \{i \in (s, ..., t)\}$, then the *expected goal-achievement* of the agent, relative to g, during the interval is the expectation

$$V^{\pi}_{\mu,g,T} \equiv E(\sum_{i=s}^{t} r_g(I_{g,s,i}))$$

where the expectation is taken over all interaction sequences $I_{g,s,i}$ drawn according to μ. We then propose

Definition 5 *The* **pragmatic general intelligence** *of an agent π, relative to the distribution ν over environments and the distribution γ over goals, is its expected performance with respect to goals drawn from γ in environments drawn from ν, over the time-scales natural to the goals; that is,*

$$\Pi(\pi) \equiv \sum_{\mu \in E, g \in \mathcal{G}, T} \nu(\mu)\gamma(g,\mu)V^{\pi}_{\mu,g,T}$$

(in those cases where this sum is convergent).

This definition formally captures the notion that "intelligence is achieving complex goals in complex environments," where "complexity" is gauged by the assumed measures ν and γ.

If ν is taken to be the universal distribution, and γ is defined to weight goals according to the universal distribution, then pragmatic general intelligence reduces to universal intelligence.

Furthermore, it is clear that a universal algorithmic agent like AIXI [Hut05] would also have a high pragmatic general intelligence, under fairly broad conditions. As the interaction history grows longer, the pragmatic general intelligence of AIXI would approach the theoretical maximum; as AIXI would implicitly infer the relevant distributions via experience. However, if significant reward discounting is involved, so that near-term rewards are weighted much higher than long-term rewards, then AIXI might compare very unfavorably in pragmatic general intelligence, to other agents designed with prior knowledge of ν, γ and τ in mind.

The most interesting case to consider is where ν and γ are taken to embody some particular bias in a real-world space of environments and goals, and this biases is appropriately reflected in the internal structure of an intelligent agent. Note that an agent needs not lack universal intelligence in order to possess pragmatic general intelligence with respect to some non-universal distribution over goals and environments. However, in general, given limited resources, there may be a tradeoff between universal intelligence and pragmatic intelligence. Which leads to the next point: how to encompass resource limitations into the definition.

One might argue that the definition of Pragmatic General Intelligence is already encompassed by Legg and Hutter's definition because one may bias the distribution of environments within the latter by considering different Turing machines underlying the Kolmogorov complexity. However this is not a general equivalence because the Solomonoff-Levin measure intrinsically decays exponentially, whereas an assumptive distribution over environments might decay at some other rate. This issue seems to merit further mathematical investigation.

7.3.4 Incorporating Computational Cost

Let $\eta_{\pi,\mu,g,T}$ be a probability distribution describing the amount of computational resources consumed by an agent π while achieving goal g over time-scale T. This is a probability distribution because we want to account for the possibility of nondeterministic agents. So, $\eta_{\pi,\mu,g,T}(Q)$ tells the probability that Q units of resources are consumed. For simplicity we amalgamate space and time resources, energetic resources, etc. into a single number Q, which is assumed to live in some subset of the positive reals. Space resources of course have to do with the size of the system's memory. Then we may define

Definition 6 *The* **efficient pragmatic general intelligence** *of an agent π with resource consumption $\eta_{\pi,\mu,g,T}$, relative to the distribution ν over environments and the distribution γ over goals, is its expected performance with respect to goals drawn from γ in environments drawn from ν, over the time-scales natural to the goals, normalized by the amount of computational effort expended to achieve each goal; that is,*

$$\Pi_{Eff}(\pi) \equiv \sum_{\mu \in E, g \in \mathcal{G}, Q, T} \frac{\nu(\mu)\gamma(g,\mu)\eta_{\pi,\mu,g,T}(Q)}{Q} V^{\pi}_{\mu,g,T}$$

(in those cases where this sum is convergent).

This is a measure that rates an agent's intelligence higher if it uses fewer computational resources to do its business. Roughly, it measures reward achieved per spacetime computation unit.

Note that, by abandoning the universal prior, we have also abandoned the proof of convergence that comes with it. In general the sums in the above definitions need not converge; and exploration of the conditions under which they do converge is a complex matter.

7.3.5 Assessing the Intelligence of Real-World Agents

The pragmatic and efficient pragmatic general intelligence measures are more "realistic" than the Legg and Hutter universal intelligence measure, in that they take into account the innate biasing and computational resource restrictions that characterize real-world intelligence. But as discussed earlier, they still live in "fantasy-land" to an extent – they gauge the intelligence of an agent via a weighted average over a wide variety of goals and environments; and they presume a simplistic relationship between agents and rewards that does not reflect the complexities of real-world cognitive architectures. It is not obvious from the foregoing how to apply these measures to real-world intelligent systems, which lack the ability to exist in such a wide variety of environments within their often brief lifespans, and mostly go about their

lives doing things other than pursuing quantified external rewards. In this brief section we describe an approach to bridging this gap. The treatment is left-semi-formal in places.

We suggest to view the definitions of pragmatic and efficient pragmatic general intelligence in terms of a "possible worlds" semantics – i.e. to view them as asking, counterfactually, how an agent *would* perform, hypothetically, on a series of tests (the tests being goals, defined in relation to environments and reward signals).

Real-world intelligent agents don't normally operate in terms of explicit goals and rewards; these are abstractions that we use to think about intelligent agents. However, this is no objection to characterizing various sorts of intelligence in terms of counterfactuals like: how would system S operate if it were trying to achieve this or that goal, in this or that environment, in order to seek reward? We can characterize various sorts of intelligence in terms of how it can be inferred an agent would perform on certain tests, even though the agent's real life does not consist of taking these tests.

This conceptual approach may seem a bit artificial, but, we don't currently see a better alternative, if one wishes to quantitatively gauge intelligence (which is, in a sense, an "artificial" thing to do in the first place). Given a real-world agent X and a mandate to assess its intelligence, the obvious alternative to looking at possible worlds in the manner of the above definitions, is just looking *directly* at the properties of the things X has achieved in the real world during its lifespan. But this isn't an easy solution, because it doesn't disambiguate which aspects of X's achievements were due to its own actions versus due to the rest of the world that X was interacting with when it made its achievements. To distinguish the amount of achievement that X "caused" via its own actions requires a model of causality, which is a complex can of worms in itself; and, critically, the standard models of causality also involve counterfactuals (asking "what would have been achieved in this situation if the agent X hadn't been there", etc.) [**?**]. Regardless of the particulars, it seems impossible to avoid counterfactual realities in assessing intelligence.

The approach we suggest – given a real-world agent X with a history of actions in a particular world, and a mandate to assess its intelligence – is to introduce an additional player, an *inference agent* δ, into the picture. The agent π modeled above is then viewed as π_X: the model of X that δ constructs, in order to explore X's inferred behaviors in various counterfactual environments. In the test situations embodied in the definitions of pragmatic and efficient pragmatic general intelligence, the environment gives π_X rewards, based on specifically configured goals. In X's real life, the relation between goals, rewards and actions will generally be significantly subtler and perhaps quite different.

We model the real world similarly to the "fantasy world" of the previous section, but with the omission of goals and rewards. We define a *naturalistic* context as one in which all goals and rewards are constant, i.e. $g_i = g_0$ and $r_i = r_0$ for all i. This is just a mathematical convention for stating that

there are no precisely-defined external goals and rewards for the agent. In a naturalistic context, we then have a situation where agents create actions based on the past history of actions and perceptions, and if there is any relevant notion of reward or goal, it is within the cognitive mechanism of some agent. A *naturalistic agent* X is then an agent π which is restricted to one particular naturalistic context, involving one particular environment μ (formally, we may achieve this within the framework of agents described above via dictating that X issues constant "null actions" a_0 in all environments except μ).

Next, we posit a metric space (Σ_μ, d) of naturalistic agents defined on a naturalistic context involving environment μ, and a subspace $\Delta \in \Sigma_\mu$ of inference agents, which are naturalistic agents that output predictions of other agents' behaviors (a notion we will not fully formalize here). If agents are represented as program trees, then d may be taken as edit distance on tree space [**?**]. Then, for each agent $\delta \in \Delta$, we may assess

- the prior probability $\theta(\delta)$ according to some assumed distribution θ
- the effectiveness $p(\delta, X)$ of δ at predicting the actions of an agent $X \in \Sigma_\mu$

We may then define

Definition 7 *The* **inference ability** *of the agent δ, relative to μ and X, is*

$$q_{\mu,X}(\delta) = \theta(\delta)\frac{\sum_{Y \in \Sigma_\mu} sim(X,Y)p(\delta,Y)}{\sum_{Y \in \Sigma_\mu} sim(X,Y)}$$

where sim is a specified decreasing function of $d(X,Y)$, such as $sim(X,Y) = \frac{1}{1+d(X,Y)}$.

To construct π_X, we may then use the model of X created by the agent $\delta \in \Delta$ with the highest inference ability relative to μ and X (using some specified ordering, in case of a tie). Having constructed π_X, we can then say that

Definition 8 *The inferred pragmatic general intelligence (relative to ν and γ) of a naturalistic agent X defined relative to an environment μ, is defined as the pragmatic general intelligence of the model π_X of X produced by the agent $\delta \in \Delta$ with maximal inference ability relative to μ (and in the case of a tie, the first of these in the ordering defined over Δ). The inferred efficient pragmatic general intelligence of X relative to μ is defined similarly.*

This provides a precise characterization of the pragmatic and efficient pragmatic intelligence of real-world systems, based on their observed behaviors. It's a bit messy; but the real world tends to be like that.

7.4 Intellectual Breadth: Quantifying the Generality of an Agent's Intelligence

We turn now to a related question: How can one quantify the degree of **generality** that an intelligent agent possesses? Above we have discussed the qualitative distinction between AGI and "Narrow AI", and intelligence as we have formalized it above is specifically intended as a measure of general intelligence. But quantifying intelligence is different than quantifying generality versus narrowness.

To make the discussion simpler, we introduce the term "context" as a shorthand for "environment/interval triple (μ, g, T)." Given a context (μ, g, T), and a set Σ of agents, one may construct a fuzzy set $Ag_{\mu,g,T}$ gathering those agents that are intelligent relative to the context; and given a set of contexts, one may also define a fuzzy set Con_π gathering those contexts with respect to which a given agent π is intelligent. The relevant formulas are:

$$\chi_{Ag_{\mu,g,T}}(\pi) = \chi_{Con_\pi}(\mu, g, T) = \frac{1}{N} \sum_Q \frac{\eta_{\mu,g,T}(Q) V^\pi_{\mu,g,T}}{Q}$$

where $N = N(\mu, g, T)$ is a normalization factor defined appropriately, e.g. via $N(\mu, g, T) = \max_\pi V^\pi_{\mu,g,T}$.

One could make similar definitions leaving out the computational cost factor Q, but we suspect that incorporating Q is a more promising direction. We then propose

Definition 9 *The* **intellectual breadth** *of an agent π, relative to the distribution ν over environments and the distribution γ over goals, is*

$$H(\chi^P_{Con_\pi}(\mu, g, T))$$

where H is the entropy and

$$\chi^P_{Con_\pi}(\mu, g, T) = \frac{\nu(\mu)\gamma(g, \mu)\chi_{Con_\pi}(\mu, g, T)}{\sum_{(\mu_\alpha, g_\beta . T_\omega)} \nu(\mu_\alpha)\gamma(g_\beta, \mu_\alpha)\chi_{Con_\pi}(\mu_\alpha, g_\beta, T_\omega)}$$

is the probability distribution formed by normalizing the fuzzy set $\chi_{Con_\pi}(\mu, g, T)$.

A similar definition of the intellectual breadth of a context (μ, g, T), relative to the distribution σ over agents, may be posited. A weakness of these definitions is that they don't try to account for dependencies between agents or contexts; perhaps more refined formulations may be developed that account explicitly for these dependencies.

Note that the intellectual breadth of an agent as defined here is largely independent of the (efficient or not) pragmatic general intelligence of that agent. One could have a rather (efficiently or not) pragmatically generally

intelligent system with little breadth: this would be a system very good at solving a fair number of hard problems, yet wholly incompetent on a larger number of hard problems. On the other hand, one could also have a terribly (efficiently or not) pragmatically generally stupid system with great intellectual breadth: i.e a system roughly equally dumb in all contexts!

Thus, one can characterize an intelligent agent as "narrow" with respect to distribution ν over environments and the distribution γ over goals, based on evaluating it as having low intellectual breadth. A "narrow AI" relative to ν and γ would then be an AI agent with a relatively high efficient pragmatic general intelligence but a relatively low intellectual breadth.

7.5 Conclusion

Our main goal in this chapter has been to push the formal understanding of intelligence in a more pragmatic direction. Much more work remains to be done, e.g. in specifying the environment, goal and efficiency distributions relevant to real-world systems, but we believe that the ideas presented here constitute nontrivial progress.

If the line of research suggested in this chapter succeeds, then eventually, one will be able to do AGI research as follows: Specify an AGI architecture formally, and then use the mathematics of general intelligence to derive interesting results about the environments, goals and hardware platforms relative to which the AGI architecture will display significant pragmatic or efficient pragmatic general intelligence, and intellectual breadth. The remaining chapters in this section present further ideas regarding how to work toward this goal. For the time being, such a mode of AGI research remains mainly for the future, but we have still found the formalism given in these chapters useful for formulating and clarifying various aspects of the CogPrime design as will be presented in later chapters.

Chapter 8
Cognitive Synergy

8.1 Cognitive Synergy

As we have seen, the formal theory of general intelligence, in its current form, doesn't really tell us much that's of use for creating real-world AGI systems. It tells us that creating extraordinarily powerful general intelligence is almost trivial if one has unrealistically huge amounts of computational resources; and that creating moderately powerful general intelligence using feasible computational resources is all about creating AI algorithms and data structures that (explicitly or implicitly) match the restrictions implied by a certain class of situations, to which the general intelligence is biased.

We've also described, in various previous chapters, some non-rigorous, conceptual principles that seem to explain key aspects of feasible general intelligence: the complementary reliance on evolution and autopoiesis, the superposition of hierarchical and heterarchical structures, and so forth. These principles can be considered as broad strategies for achieving general intelligence in certain broad classes of situations. Although, a lot of research needs to be done to figure out nice ways to describe, for instance, in what class of situations evolution is an effective learning strategy, in what class of situations dual hierarchical/heterarchical structure is an effective way to organize memory, etc.

In this chapter we'll dig deeper into one of the "general principle of feasible general intelligences" briefly alluded to earlier: the *cognitive synergy* principle, which is both a conceptual hypothesis about the structure of generally intelligent systems in certain classes of environments, and a design principle used to guide the architecting of CogPrime .

We will focus here on cognitive synergy specifically in the case of "multi-memory systems," which we define as intelligent systems (like CogPrime) whose combination of environment, embodiment and motivational system make it important for them to possess memories that divide into partially but not wholly distinct components corresponding to the categories of:

- Declarative memory
- Procedural memory (memory about how to do certain things)
- Sensory and episodic memory
- Attentional memory (knowledge about what to pay attention to in what contexts
- Intentional memory (knowledge about the system's own goals and subgoals)

In Chapter 9 below we present a detailed argument as to how the requirement for a multi-memory underpinning for general intelligence emerges from certain underlying assumptions regarding the measurement of the simplicity of goals and environments; but the points made here do not rely on that argument. What they do rely on is the assumption that, in the intelligence in question, the different components of memory are significantly but not wholly distinct. That is, there are significant "family resemblances" between the memories of a single type, yet there are also thoroughgoing connections between memories of different types.

The cognitive synergy principle, if correct, applies to any AI system demonstrating intelligence in the context of embodied, social communication. However, one may also take the theory as an explicit guide for constructing AGI systems; and of course, the bulk of this book describes one AGI architecture, CogPrime , designed in such a way.

It is possible to cast these notions in mathematical form, and we make some efforts in this direction in Appendix B, using the languages of category theory and information geometry. However, this formalization has not yet led to any rigorous proof of the generality of cognitive synergy nor any other exciting theorems; with luck this will come as the mathematics is further developed. In this chapter the presentation is kept on the heuristic level, which is all that is critically needed for motivating the CogPrime design.

8.2 Cognitive Synergy

The essential idea of cognitive synergy, in the context of multi-memory systems, may be expressed in terms of the following points:

1. Intelligence, relative to a certain set of environments, may be understood as the capability to achieve complex goals in these environments.
2. With respect to certain classes of goals and environments (see Chapter 9 for a hypothesis in this regard), an intelligent system requires a "multi-memory" architecture, meaning the possession of a number of specialized yet interconnected knowledge types, including: declarative, procedural, attentional, sensory, episodic and intentional (goal-related). These knowledge types may be viewed as different sorts of pattern that a system recognizes in itself and its environment. Knowledge of these various different

types must be interlinked, and in some cases may represent differing views of the same content (see Figure 8.2)
3. Such a system must possess knowledge creation (i.e. pattern recognition / formation) mechanisms corresponding to each of these memory types. These mechanisms are also called "cognitive processes."
4. Each of these cognitive processes, to be effective, must have the capability to recognize when it lacks the information to perform effectively on its own; and in this case, to dynamically and interactively draw information from knowledge creation mechanisms dealing with other types of knowledge
5. This cross-mechanism interaction must have the result of enabling the knowledge creation mechanisms to perform much more effectively in combination than they would if operated non-interactively. This is "cognitive synergy."

While these points are implicit in the theory of mind given in [Goe06a], they are not articulated in this specific form there.

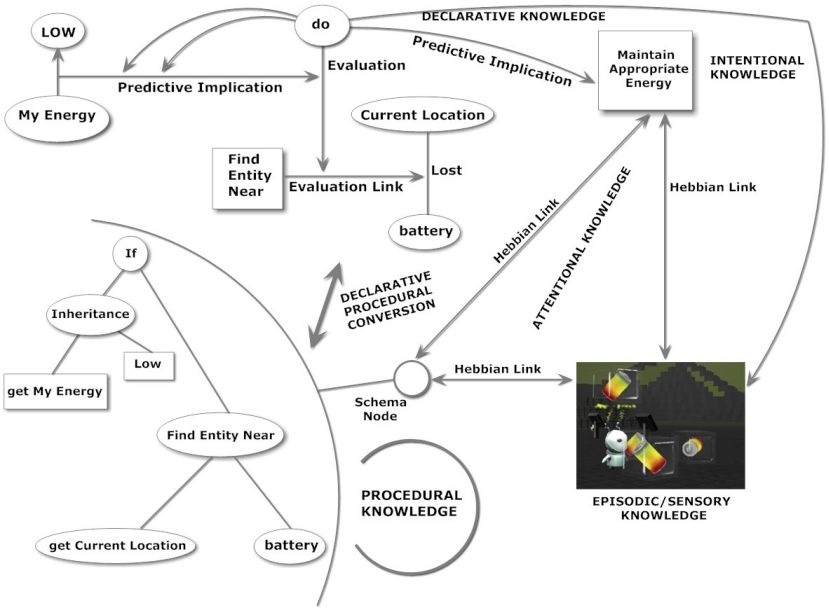


Fig. 8.1 Illustrative example of the interactions between multiple types of knowledge, in representing a simple piece of knowledge. Generally speaking, one type of knowledge can be converted to another, at the cost of some loss of information. The synergy between cognitive processes associated with corresponding pieces of knowledge, possessing different type, is a critical aspect of general intelligence.

Interactions as mentioned in Points 4 and 5 in the above list are the real conceptual meat of the cognitive synergy idea. One way to express the key idea here is that most AI algorithms suffer from combinatorial explosions: the number of possible elements to be combined in a synthesis or analysis is just too great, and the algorithms are unable to filter through all the possibilities, given the lack of intrinsic constraint that comes along with a "general intelligence" context (as opposed to a narrow-AI problem like chess-playing, where the context is constrained and hence restricts the scope of possible combinations that needs to be considered). In an AGI architecture based on cognitive synergy, the different learning mechanisms must be designed specifically to interact in such a way as to palliate each others' combinatorial explosions - so that, for instance, each learning mechanism dealing with a certain sort of knowledge, must synergize with learning mechanisms dealing with the other sorts of knowledge, in a way that decreases the severity of combinatorial explosion.

One prerequisite for cognitive synergy to work is that each learning mechanism must recognize when it is "stuck," meaning it's in a situation where it has inadequate information to make a confident judgment about what steps to take next. Then, when it does recognize that it's stuck, it may request help from other, complementary cognitive mechanisms.

A theoretical notion closely related to cognitive synergy is the *cognitive schematic*, formalized in Chapter 7 above, which states that the activity of the different cognitive processes involved in an intelligent system may be modeled in terms of the schematic implication

$$Context \wedge Procedure \rightarrow Goal$$

where the Context involves sensory, episodic and/or declarative knowledge; and attentional knowledge is used to regulate how much resource is given to each such schematic implication in memory. Synergy among the learning processes dealing with the context, the procedure and the goal is critical to the adequate execution of the cognitive schematic using feasible computational resources.

Finally, drilling a little deeper into Point 3 above, one arrives at a number of possible knowledge creation mechanisms (cognitive processes) corresponding to each of the key types of knowledge. Figure 8.2 below gives a high-level overview of the main types of cognitive process considered in the current version of Cognitive Synergy Theory, categorized according to the type of knowledge with which each process deals.

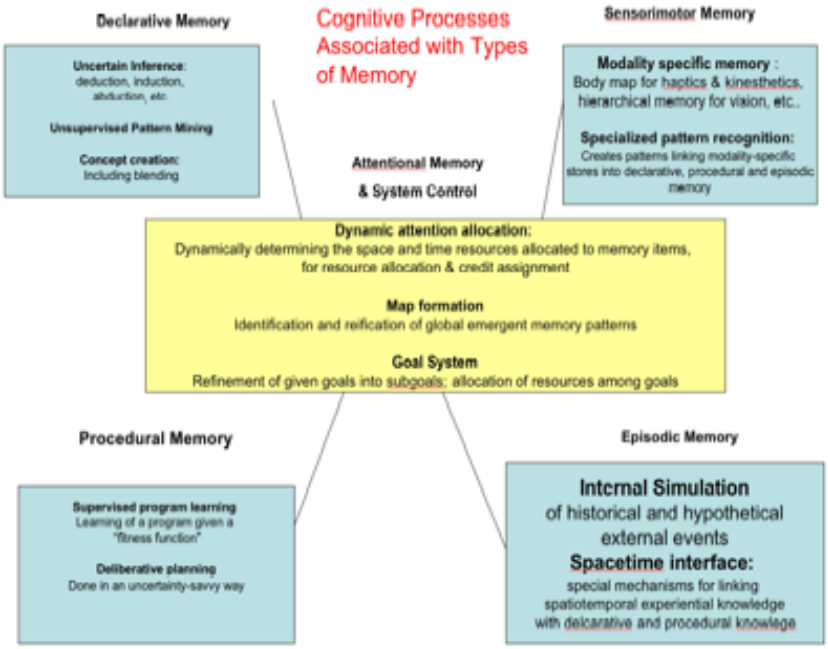


Fig. 8.2 High-level overview of the key cognitive dynamics considered here in the context of cognitive synergy. The cognitive synergy principle describes the behavior of a system as it pursues a set of goals (which in most cases may be assumed to be supplied to the system "a priori", but then refined by inference and other processes). The assumed intelligent agent model is roughly as follows: At each time the system chooses a set of procedures to execute, based on its judgments regarding which procedures will best help it achieve its goals in the current context. These procedures may involve external actions (e.g. involving conversation, or controlling an agent in a simulated world) and/or internal cognitive actions. In order to make these judgments it must effectively manage declarative, procedural, episodic, sensory and attentional memory, each of which is associated with specific algorithms and structures as depicted in the diagram. There are also global processes spanning all the forms of memory, including the allocation of attention to different memory items and cognitive processes, and the identification and reification of system-wide activity patterns (the latter referred to as "map formation")

8.3 Cognitive Synergy in CogPrime

Different cognitive systems will use different processes to fulfill the various roles identified in Figure **??** above. Here we briefly preview the basic cognitive processes that the CogPrime AGI design uses for these roles, and the synergies that exist between these.

8.3.1 Cognitive Processes in CogPrime

: a Cognitive Synergy Based Architecture..." from ICCI 2009

Table 8.1 default

|Table will go here|

Table 8.2 The OpenCogPrime data structures used to represent the key knowledge types involved

Table 8.3 default

|Table will go here|

Table 8.4 Key cognitive processes, and the algorithms that play their roles in CogPrime

Table 8.5 default

|Table will go here|

Table 8.6 Key OpenCogPrime cognitive processes categorized according to knowledge type and process type

Tables 8.1 and 8.3 present the key structures and processes involved in CogPrime , identifying each one with a certain memory/process type as considered in cognitive synergy theory. That is: each of these cognitive structures or processes deals with one or more types of memory – declarative, procedural, sensory, episodic or attentional. Table 8.5 describes the key CogPrime processes in terms of the "analysis vs. synthesis" distinction. Finally, Tables **??** and **??** exemplify these structures and processes in the context of embodied virtual agent control.

In the CogPrime context, a procedure in this cognitive schematic is a program tree stored in the system's procedural knowledge base; and a context is a (fuzzy, probabilistic) logical predicate stored in the AtomSpace, that holds, to a certain extent, during each interval of time. A goal is a fuzzy logical predicate that has a certain value at each interval of time, as well.

Attentional knowledge is handled in CogPrime by the ECAN artificial economics mechanism, that continually updates ShortTermImportance and LongTerm Importance values associated with each item in the CogPrime system's memory, which control the amount of attention other cognitive mechanisms pay to the item, and how much motive the system has to keep the

item in memory. HebbianLinks are then created between knowledge items that often possess ShortTermImportance at the same time; this is CogPrime 's version of traditional Hebbian learning.

ECAN has deep interactions with other cognitive mechanisms as well, which are essential to its efficient operation; for instance, PLN inference may be used to help ECAN extrapolate conclusions about what is worth paying attention to, and MOSES may be used to recognize subtle attentional patterns. ECAN also handles "assignment of credit", the figuring-out of the causes of an instance of successful goal-achievement, drawing on PLN and MOSES as needed when the causal inference involved here becomes difficult.

The synergies between CogPrime 's cognitive processes are well summarized in Table 6 below, which is a 16x16 matrix summarizing a host of interprocess interactions generic to CST.

One key aspect of how CogPrime implements cognitive synergy is PLN's sophisticated management of the confidence of judgments. This ties in with the way OpenCogPrimes PLN inference framework represents truth values in terms of multiple components (as opposed to the single probability values used in many probabilistic inference systems and formalisms): each item in OpenCogPrimes declarative memory has a confidence value associated with it, which tells how much weight the system places on its knowledge about that memory item. This assists with cognitive synergy as follows: A learning mechanism may consider itself "stuck", generally speaking, when it has no high-confidence estimates about the next step it should take.

Without reasonably accurate confidence assessment to guide it, intercomponent interaction could easily lead to increased rather than decreased combinatorial explosion. And of course there is an added recursion here, in that confidence assessment is carried out partly via PLN inference, which in itself relies upon these same synergies for its effective operation.

To illustrate this point further, consider one of the synergetic aspects described in 8.4 below: the role cognitive synergy plays in deductive inference. Deductive inference is a hard problem in general - but what is hard about it is not carrying out inference steps, but rather "inference control" (i.e., choosing which inference steps to carry out). Specifically, what must happen for deduction to succeed in CogPrime is:

1. the system must recognize when its deductive inference process is "stuck", i.e. when the PLN inference control mechanism carrying out deduction has no clear idea regarding which inference step(s) to take next, even after considering all the domain knowledge at is disposal
2. in this case, the system must defer to another learning mechanism to gather more information about the different choices available - and the other learning mechanism chosen must, a reasonable percentage of the time, actually provide useful information that helps PLN to get "unstuck" and continue the deductive process

For instance, deduction might defer to the "attentional knowledge" subsystem, and make a judgment as to which of the many possible next deductive steps are most associated with the goal of inference and the inference steps taken so far, according to the HebbianLinks constructed by the attention allocation subsystem, based on observed associations. Or, if this fails, deduction might ask MOSES (running in supervised categorization mode) to learn predicates characterizing some of the terms involving the possible next inference steps. Once MOSES provides these new predicates, deduction can then attempt to incorporate these into its inference process, hopefully (though not necessarily) arriving at a higher-confidence next step..

8.4 Some Critical Synergies

Referring back to Figure 8.2, and summarizing many of the ideas in the previous section, Table 8.4 enumerates a number of specific ways in which the cognitive processes mentioned in the Figure may synergize with one another, potentially achieving dramatically greater efficiency than would be possible on their own.

Of course, realizing these synergies on the practical algorithmic level requires significant inventiveness and may be approached in many different ways. The specifics of how CogPrime manifests these synergies are discussed in many following chapters.

8.5 The Cognitive Schematic

Now we return to the "cognitive schematic" notion, according to which various cognitive processes involved in intelligence may be understood to work together via the implication

$$Context \wedge Procedure \rightarrow Goal < p >$$

(summarized $C \wedge P \rightarrow G$). Semi-formally, this implication may interpreted to mean: "If the context C appears to hold currently, then if I enact the procedure P, I can expect to achieve the goal G with certainty p."

The cognitive schematic leads to a conceptualization of the internal action of an intelligent system as involving two key categories of learning:

- **Analysis**: Estimating the probability p of a posited $C \wedge P \rightarrow G$ relationship
- **Synthesis**: Filling in one or two of the variables in the cognitive schematic, given assumptions regarding the remaining variables, and directed by the goal of maximizing the probability of the cognitive schematic

How ---> Helps | |/	Map formation	Goal system	Simulation	Sensorimotor pattern recognition
Uncertain inference	Creates new concepts and relationships, enabling briefer useful inference trails	Goal refinement enables more careful goal-based inference pruning	- Simulations provide a method of testing speculative inferential conclusions - Simulations suggest hypotheses to be explored via inference	Creates new concepts and relationships, enabling briefer useful inference trails
Supervised procedure learning	Creates new procedures to be used as modules in candidate procedures	Goal refinement allows more precise definition of fitness functions, making procedure learning's job easier	Simulation provides a method of "fitness estimation" allowing inexpensive testing of candidate procedures	Extraction of sensorimotor patterns allows creation of abstracted fitness functions for (inferentially and simulatively) evaluating procedures guiding real-world actions
Attention allocation	Creates new concepts grouping "attentionally related" memory items, enabling AA to find subtler attentional patterns involving these nodes	Goal refinement allows more accurately goal-driven allocation of attention	Simulation provides data for attention allocation -- allowing attentional information to be extracted from co-occurences observed in simulation	Creates concepts grouping "attentionally related" memory items, enabling AA to find subtler attentional patterns involving these nodes
Concept creation	Creates new concepts to be fed into other concept creation mechanisms	Goal refinement provides more precise definition of criteria via which new concepts are created	Utility of concepts may be assessed via creating simulated entities embodying the new concepts and seeing what they lead to in simulation	Creates new concepts to be fed into other concept creation mechanisms

Fig. 8.3 This table, and the following ones, show some of the synergies between the primary cognitive processes explicitly used in CogPrime .

How ---> Helps | |/	Uncertain inference	Supervised procedure learning	Attention allocation	Concept creation
Uncertain inference	NA	When inference gets stuck in an inference trail, it can ask procedure learning to learn new patterns regarding concepts in the inference trail (if there is adequate data regarding the concepts)	Importance levels allow pruning of inference trees	Provides new concepts, allowing briefer useful inference trails
Supervised procedure learning	Inference can be used to allow prior experience to guide each instance of procedure learning.	NA	Importance levels may be used to bias choices made in the course of procedure learning (e.g. in OCP, in the fitness evaluation and representation-building phases of MOSES)	Provides new concepts, allowing compacter programs using new concepts in various roles
Attention allocation	Enables inference of new HebbianLinks and HebbianPredicates from existing ones	Procedure learning can recognize patterns in historical system activity, which are then used to build concepts and relationships guiding attention allocation	NA	Combination of concepts formed via map formation, may lead to new concepts that even better direct attention
Concept creation	Allows inferential assessment of the value of new concepts	Procedure learning can be used to search for high-quality blends of existing concepts (using e.g. inferential and attentional knowledge in the fitness functions)	Allows assessment of the value of new concepts based on historical attentional knowledge	NA

How ---> Helps | |/	**Uncertain inference**	**Supervised procedure learning**	**Attention allocation**	**Concept creation**
Map formation	Speculative inference can help map formation guess which maps to hunt for	Procedure learning can be used to search for maps that are more complex than mere "co-occurrence"	Attention allocation provides the raw data for map formation	*No significant direct synergy*
Goal system	Inference can carry out goal refinement	*No significant direct synergy*	Flow of importance among subgoals determines which subgoals get used, versus being forgotten	Concept creation can be used to provide raw data for goal refinement (e.g. a new subgoal that blends two others)
Simulation	In order to provide data for setting up simulations, inference will often be needed	*No significant direct synergy*	Attention allocation tells which portions of a simulation need to be run in more detail	*No significant direct synergy*
Sensorimotor pattern recognition	Speculative inference helps fill in gaps in sensory data	Procedure learning can be used to find subtle patterns in sensorimotor data	Attention allocation guides pattern recognition via indicating which sensorimotor stimuli and patterns tend to be associatively linked	New concepts may be created that then are found to serve as significant patterns in sensorimotor data

How ---> Helps | |/	**Map formation**	**Goal system**	**Simulation**	**Sensorimotor pattern recognition**
Map formation	NA	Map formation may focus on finding maps related to subgoals, and good subgoal refinement helps here	*No significant direct synergy*	*No significant direct synergy*
Goal system	Concepts formed from maps may be useful raw material for forming subgoals	NA	*No significant direct synergy*	*No significant direct synergy*
Simulation	*No significant direct synergy*	*No significant direct synergy*	NA	Presence of recognized sensorimotor patterns may be used to judge whether a simulation is sufficiently accurate
Sensorimotor pattern recognition	Concepts formed from maps may usefully guide sensorimotor pattern search	Directing pattern search toward patterns pertinent to subgoals, may make the task far easier	Patterns recognized in simulations may then be checked for presence in real sensorimotor data	NA

More specifically, where synthesis is concerned, some key examples are:

- The MOSES probabilistic evolutionary program learning algorithm is applied to find P, given fixed C and G. Internal simulation is also used, for the purpose of creating a simulation embodying C and seeing which P lead to the simulated achievement of G.
  - *Example: A virtual dog learns a procedure P to please its owner (the goal G) in the context C where there is a ball or stick present and the owner is saying "fetch".*
- PLN inference, acting on declarative knowledge, is used for choosing C, given fixed P and G (also incorporating sensory and episodic knowledge as appropriatel). Simulation may also be used for this purpose.
  - *Example: A virtual dog wants to achieve the goal G of getting food, and it knows that the procedure P of begging has been successful at this before, so it seeks a context C where begging can be expected to get it food. Probably this will be a context involving a friendly person.*
- PLN-based goal refinement is used to create new subgoals G to sit on the right hand side of instances of the cognitive schematic.
  - *Example: Given that a virtual dog has a goal of finding food, it may learn a subgoal of following other dogs, due to observing that other dogs are often heading toward their food.*
- Concept formation heuristics are used for choosing G and for fueling goal refinement, but especially for choosing C (via providing new candidates for C). They are also used for choosing P, via a process called "predicate schematization" that turns logical predicates (declarative knowledge) into procedures.
  - *Example: At first a virtual dog may have a hard time predicting which other dogs are going to be mean to it. But it may eventually observe common features among a number of mean dogs, and thus form its own concept of "pit bull," without anyone ever teaching it this concept explicitly.*

Where analysis is concerned:

- PLN inference, acting on declarative knowledge, is used for estimating the probability of the implication in the cognitive schematic, given fixed C, P and G. Episodic knowledge is also used this regard, via enabling estimation of the probability via simple similarity matching against past experience. Simulation is also used: multiple simulations may be run, and statistics may be captured therefrom.
  - *Example: To estimate the degree to which asking Bob for food (the procedure P is "asking for food", the context C is "being with Bob")*

will achieve the goal G of getting food, the virtual dog may study its memory to see what happened on previous occasions where it or other dogs asked Bob for food or other things, and then integrate the evidence from these occasions.

- Procedural knowledge, mapped into declarative knowledge and then acted on by PLN inference, can be useful for estimating the probability of the implication $C \wedge P \rightarrow G$, in cases where the probability of $C \wedge P_1 \rightarrow G$ is known for some P_1 related to P.
  - *Example: knowledge of the internal similarity between the procedure of asking for food and the procedure of asking for toys, allows the virtual dog to reason that if asking Bob for toys has been successful, maybe asking Bob for food will be successful too.*
- Inference, acting on declarative or sensory knowledge, can be useful for estimating the probability of the implication $C \wedge P \rightarrow G$, in cases where the probability of $C_1 \wedge P \rightarrow G$ is known for some C_1 related to C.
  - *Example: if Bob and Jim have a lot of features in common, and Bob often responds positively when asked for food, then maybe Jim will too.*
- Inference can be used similarly for estimating the probability of the implication $C \wedge P \rightarrow G$, in cases where the probability of $C \wedge P \rightarrow G_1$ is known for some G_1 related to G. Concept creation can be useful indirectly in calculating these probability estimates, via providing new concepts that can be used to make useful inference trails more compact and hence easier to construct.
  - *Example: The dog may reason that because Jack likes to play, and Jack and Jill are both children, maybe Jill likes to play too. It can carry out this reasoning only if its concept creation process has invented the concept of "child" via analysis of observed data.*

In these examples we have focused on cases where two terms in the cognitive schematic are fixed and the third must be filled in; but just as often, the situation is that only one of the terms is fixed. For instance, if we fix G, sometimes the best approach will be to collectively learn C and P. This requires either a procedure learning method that works interactively with a declarative-knowledge-focused concept learning or reasoning method; or a declarative learning method that works interactively with a procedure learning method. That is, it requires the sort of cognitive synergy built into the CogPrime design.

8.6 Cognitive Synergy for Procedural and Declarative Learning

We now present a little more algorithmic detail regarding the operation and synergetic interaction of CogPrime 's two most sophisticated components: the MOSES procedure learning algorithm (see Chapter 33), and the PLN uncertain inference framework (see Chapter 34). The treatment is necessarily quite compact, since we have not yet reviewed the details of either MOSES or PLN; but as well as illustrating the notion of cognitive synergy more concretely, perhaps the high-level discussion here will make clearer how MOSES and PLN fit into the big picture of CogPrime .

8.6.1 Cognitive Synergy in MOSES

MOSES, CogPrime 's primary algorithm for learning procedural knowledge, has been tested on a variety of application problems including standard GP test problems, virtual agent control, biological data analysis and text classification [Loo06]. It represents procedures internally as program trees. Each node in a MOSES program tree is supplied with a "knob," comprising a set of values that may potentially be chosen to replace the data item or operator at that node. So for instance a node containing the number 7 may be supplied with a knob that can take on any integer value. A node containing a while loop may be supplied with a knob that can take on various possible control flow operators including conditionals or the identity. A node containing a procedure representing a particular robot movement, may be supplied with a knob that can take on values corresponding to multiple possible movements. Following a metaphor suggested by Douglas Hofstadter [**?**], MOSES learning covers both "knob twiddling" (setting the values of knobs) and "knob creation."

MOSES is invoked within CogPrime in a number of ways, but most commonly for finding a procedure P satisfying a probabilistic implication $C\&P \rightarrow G$ as described above, where C is an observed context and G is a system goal. In this case the probability value of the implication provides the "scoring function" that MOSES uses to assess the quality of candidate procedures.

For example, suppose an CogPrime -controlled robot is trying to learn to play the game of "tag." (I.e. a multi-agent game in which one agent is specially labeled "it", and runs after the other player agents, trying to touch them. Once another agent is touched, it becomes the new "it" and the previous "it" becomes just another player agent.) Then its context C is that others are trying to play a game they call "tag" with it; and we may assume its goals are to please them and itself, and that it has figured out that in order to achieve this

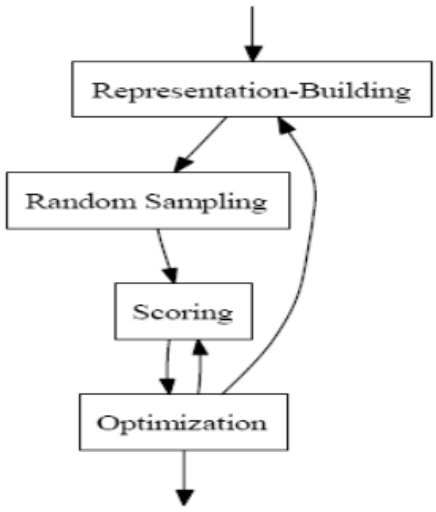


Fig. 8.4 High-Level Control Flow of MOSES Algorithm

goal it should learn some procedure to follow when interacting with others who have said they are playing "tag." In this case a potential tag-playing procedure might contain nodes for physical actions like $step\_forward(speed\ s)$, as well as control flow nodes containing operators like $ifelse$ (for instance, there would probably be a conditional telling the robot to do something different depending on whether someone seems to be chasing it). Each of these program tree nodes would have an appropriate knob assigned to it. And the scoring function would evaluate a procedure P in terms of how successfully the robot played tag when controlling its behaviors according to P (noting that it may also be using other control procedures concurrently with P). It's worth noting here that evaluating the scoring function in this case involves some inference already, because in order to tell if it is playing tag successfully, in a real-world context, it must watch and understand the behavior of the other players.

MOSES follows the high-level control flow depicted in Figure 33.1, which corresponds to the following process for evolving a metapopulation of "demes" of programs (each deme being a set of relatively similar programs, forming a sort of island in program space):

1. Construct an initial set of knobs based on some prior (e.g., based on an empty program; or more interestingly, using prior knowledge **supplied by PLN inference** based on the system's memory) and use it to generate an initial random sampling of programs. Add this deme to the metapopulation.
2. Select a deme from the metapopulation and update its sample, as follows:

   a. Select some promising programs from the deme's existing sample to use for modeling, according to the scoring function.
   b. Considering the promising programs as collections of knob settings, generate new collections of knob settings by applying some (competent) optimization algorithm. For best performance on difficult problems, it is important to use an optimization algorithm that makes use of the system's memory in its choices, **consulting PLN inference** to help estimate which collections of knob settings will work best.
   c. Convert the new collections of knob settings into their corresponding programs, reduce the programs to normal form, evaluate their scores, and integrate them into the deme's sample, replacing less promising programs. In the case that scoring is expensive, score evaluation may be preceded by score estimation, which may use **PLN inference**, enaction of procedures in an **internal simulation environment**, and/or similarity matching against **episodic memory**.

3. For each new program that meet the criterion for creating a new deme, if any:

   a. Construct a new set of knobs (a process called "representation-building") to define a region centered around the program (the deme's *exemplar)*, and use it to generate a *new* random sampling of programs, producing a new deme.
   b. Integrate the new deme into the metapopulation, possibly displacing less promising demes.

4. Repeat from step 2.

MOSES is a complex algorithm and each part plays its role; if any one part is removed the performance suffers significantly [Loo06]. However, the main point we want to highlight here is the role played by synergetic interactions between MOSES and other cognitive components such as PLN, simulation and episodic memory, as indicated in **boldface** in the above pseudocode. MOSES is a powerful procedure learning algorithm, but used on its own it runs into scalability problems like any other such algorithm; the reason we feel it has potential to play a major role in a human-level AI system is its capacity for productive interoperation with other cognitive components.

Continuing the "tag" example, the power of MOSES's integration with other cognitive processes would come into play if, before learning to play tag, the robot has already played simpler games involving chasing. If the robot already has experience chasing and being chased by other agents, then its episodic and declarative memory will contain knowledge about how to pursue and avoid other agents in the context of running around an environment full of objects, and this knowledge will be deployable within the appropriate parts of MOSES's Steps 1 and 2. Cross-process and cross-memory-type integration make it tractable for MOSES to act as a "transfer learning" algorithm, not just a task-specific machine-learning algorithm.

8.6.2 Cognitive Synergy in PLN

While MOSES handles much of CogPrime 's procedural learning, and OpenCogPrimes internal simulation engine handles most episodic knowledge, CogPrime 's primary tool for handling declarative knowledge is an uncertain inference framework called Probabilistic Logic Networks (PLN). The complexities of PLN are the topic of a lengthy technical monograph [GMIH08], and here we will eschew most details and focus mainly on pointing out how PLN seeks to achieve efficient inference control via integration with other cognitive processes.

As a logic, PLN is broadly integrative: it combines certain term logic rules with more standard predicate logic rules, and utilizes both fuzzy truth values and a variant of imprecise probabilities called *indefinite probabilities*. PLN mathematics tells how these uncertain truth values propagate through its logic rules, so that uncertain premises give rise to conclusions with reasonably accurately estimated uncertainty values. This careful management of uncertainty is critical for the application of logical inference in the robotics context, where most knowledge is abstracted from experience and is hence highly uncertain.

PLN can be used in either forward or backward chaining mode; and in the language introduced above, it can be used for either analysis or synthesis. As an example, we will consider backward chaining analysis, exemplified by the problem of a robot preschool-student trying to determine whether a new playmate "Bob" is likely to be a regular visitor to is preschool or not (evaluating the truth value of the implication $Bob \rightarrow regular\_visitor$). The basic backward chaining process for PLN analysis looks like:

1. Given an implication $L \equiv A \rightarrow B$ whose truth value must be estimated (for instance $L \equiv C\&P \rightarrow G$ as discussed above), create a list $(A_1, ..., A_n)$ of *(inference rule, stored knowledge)* pairs that might be used to produce L
2. Using analogical reasoning to prior inferences, assign each A_i a probability of success
   - If some of the A_i are estimated to have reasonable probability of success at generating reasonably confident estimates of L's truth value, then invoke Step 1 with A_i in place of L (at this point the inference process becomes recursive)
   - If none of the A_i looks sufficiently likely to succeed, then inference has "gotten stuck" and another cognitive process should be invoked, e.g.
     - **Concept creation** may be used to infer new concepts related to A and B, and then Step 1 may be revisited, in the hope of finding a new, more promising A_i involving one of the new concepts
     - **MOSES** may be invoked with one of several special goals, e.g. the goal of finding a procedure P so that $P(X)$ predicts whether

$X \to B$. If MOSES finds such a procedure P then this can be converted to declarative knowledge understandable by PLN and Step 1 may be revisited....
- **Simulations** may be run in CogPrime 's internal simulation engine, so as to observe the truth value of $A \to B$ in the simulations; and then Step 1 may be revisited....

The combinatorial explosion of inference control is combatted by the capability to defer to other cognitive processes when the inference control procedure is unable to make a sufficiently confident choice of which inference steps to take next. Note that just as MOSES may rely on PLN to model its evolving populations of procedures, PLN may rely on MOSES to create complex knowledge about the terms in its logical implications. This is just one example of the multiple ways in which the different cognitive processes in CogPrime interact synergetically; a more thorough treatment of these interactions is given in Chapter 48.

In the "new playmate" example, the interesting case is where the robot initially seems not to know enough about Bob to make a solid inferential judgment (so that none of the A_i seem particularly promising). For instance, it might carry out a number of possible inferences and not come to any reasonably confident conclusion, so that the reason none of the A_i seem promising is that all the decent-looking ones have been tried already. So it might then recourse to MOSES, simulation or concept creation.

For instance, the PLN controller could make a list of everyone who has been a regular visitor, and everyone who has not been, and pose MOSES the task of figuring out a procedure for distinguishing these two categories. This procedure could then used directly to make the needed assessment, or else be translated into logical rules to be used within PLN inference. For example, perhaps MOSES would discover that older males wearing ties tend not to become regular visitors. If the new playmate is an older male wearing a tie, this is directly applicable. But if the current playmate is wearing a tuxedo, then PLN may be helpful via reasoning that even though a tuxedo is not a tie, it's a similar form of fancy dress – so PLN may extend the MOSES-learned rule to the present case and infer that the new playmate is not likely to be a regular visitor.

8.7 Is Cognitive Synergy Tricky?

[1]

In this section we use the notion of cognitive synergy to explore a question that arises frequently in the AGI community: the well-known difficulty of measuring intermediate progress toward human-level AGI. We explore some

[1] This section co-authored with Jared Wigmore

potential reasons underlying this, via extending the notion of cognitive synergy to a more refined notion of "tricky cognitive synergy." These ideas are particularly relevant to the problem of creating a roadmap toward AGI, as we'll explore in Chapter 17 below.

8.7.1 The Puzzle: Why Is It So Hard to Measure Partial Progress Toward Human-Level AGI?

It's not entirely straightforward to create tests to measure the *final achievement* of human-level AGI, but there are some fairly obvious candidates here. There's the Turing Test (fooling judges into believing you're human, in a text chat) the video Turing Test, the Robot College Student test (passing university, via being judged exactly the same way a human student would), etc. There's certainly no agreement on which is the most meaningful such goal to strive for, but there's broad agreement that a number of goals of this nature basically make sense.

On the other hand, how does one measure whether one is, say, 50 percent of the way to human-level AGI? Or, say, 75 or 25 percent?

It's possible to pose many "practical tests" of incremental progress toward human-level AGI, with the property that IF a proto-AGI system passes the test using a certain sort of architecture and/or dynamics, then this implies a certain amount of progress toward human-level AGI *based on particular theoretical assumptions about AGI*. However, in each case of such a practical test, it seems intuitively likely *to a significant percentage of AGI researchers* that there is some way to "game" the test via designing a system specifically oriented toward passing that test, and which doesn't constitute dramatic progress toward AGI.

Some examples of practical tests of this nature would be

- The Wozniak "coffee test": go into an average American house and figure out how to make coffee, including identifying the coffee machine, figuring out what the buttons do, finding the coffee in the cabinet, etc.
- Story understanding – reading a story, or watching it on video, and then answering questions about what happened (including questions at various levels of abstraction)
- Graduating (virtual-world or robotic) preschool
- Passing the elementary school reading curriculum (which involves reading and answering questions about some picture books as well as purely textual ones)
- Learning to play an arbitrary video game based on experience only, or based on experience plus reading instructions

One interesting point about tests like this is that each of them seems to *some* AGI researchers to encapsulate the crux of the AGI problem, and be

unsolvable by any system not far along the path to human-level AGI – yet seems to other AGI researchers, with different conceptual perspectives, to be something probably game-able by narrow-AI methods. And of course, given the current state of science, there's no way to tell which of these practical tests really can be solved via a narrow-AI approach, except by having a lot of people try really hard over a long period of time.

A question raised by these observations is whether there is some *fundamental reason* why it's hard to make an objective, theory-independent measure of intermediate progress toward advanced AGI. Is it just that we haven't been smart enough to figure out the right test – or is there some conceptual reason why the very notion of such a test is problematic?

We don't claim to know for sure – but in the rest of this section we'll outline one possible reason why the latter might be the case.

8.7.2 A Possible Answer: Cognitive Synergy is Tricky!

Why might a solid, objective empirical test for intermediate progress toward AGI be an infeasible notion? One possible reason, we suggest, is precisely *cognitive synergy*, as discussed above.

The cognitive synergy hypothesis, in its simplest form, states that human-level AGI intrinsically depends on the synergetic interaction of multiple components (for instance, as in the OpenCog design, multiple memory systems each supplied with its own learning process). In this hypothesis, for instance, it might be that there are 10 critical components required for a human-level AGI system. Having all 10 of them in place results in human-level AGI, but having only 8 of them in place results in having a dramatically impaired system – and maybe having only 6 or 7 of them in place results in a system that can hardly do anything at all.

Of course, the reality is almost surely not as strict as the simplified example in the above paragraph suggests. No AGI theorist has really posited a list of 10 crisply-defined subsystems and claimed them necessary and sufficient for AGI. We suspect there are many different routes to AGI, involving integration of different sorts of subsystems. However, if the cognitive synergy hypothesis is correct, then human-level AGI behaves *roughly* like the simplistic example in the prior paragraph suggests. Perhaps instead of using the 10 components, you could achieve human-level AGI with 7 components, but having only 5 of these 7 would yield drastically impaired functionality – etc. Or the point could be made without any decomposition into a finite set of components, using continuous probability distributions. To mathematically formalize the cognitive synergy hypothesis becomes complex, but here we're only aiming for a qualitative argument. So for illustrative purposes, we'll stick with the "10 components" example, just for communicative simplicity.

Next, let's suppose that for any given task, there are ways to achieve this task using a system that is much simpler than any subset of size 6 drawn from the set of 10 components needed for human-level AGI, but works much better for the task than this subset of 6 components(assuming the latter are used as a set of only 6 components, without the other 4 components).

Note that this supposition is a good bit stronger than mere cognitive synergy. For lack of a better name, we'll call it *tricky cognitive synergy*. The tricky cognitive synergy hypothesis would be true if, for example, the following possibilities were true:

- creating components to serve as parts of a synergetic AGI is *harder* than creating components intended to serve as parts of simpler AI systems without synergetic dynamics
- components capable of serving as parts of a synergetic AGI are necessarily *more complicated* than components intended to serve as parts of simpler AGI systems.

These certainly seem reasonable possibilities, since to serve as a component of a synergetic AGI system, a component must have the internal flexibility to usefully handle interactions with a lot of other components as well as to solve the problems that come its way. In a CogPrime context, these possibilities ring true, in the sense that tailoring an AI process for tight integration with other AI processes within CogPrime , tends to require more work than preparing a conceptually similar AI process for use on its own or in a more task-specific narrow AI system.

It seems fairly obvious that, if tricky cognitive synergy really holds up as a property of human-level general intelligence, the difficulty of formulating tests for intermediate progress toward human-level AGI follows as a consequence. Because, according to the tricky cognitive synergy hypothesis, any test is going to be more easily solved by some simpler narrow AI process than by a *partially complete* human-level AGI system.

8.7.3 Conclusion

We haven't proved anything here, only made some qualitative arguments. However, these arguments do seem to give a plausible explanation for the empirical observation that positing tests for intermediate progress toward human-level AGI is a very difficult prospect. If the theoretical notions sketched here are correct, then this difficulty is not due to incompetence or lack of imagination on the part of the AGI community, nor due to the primitive state of the AGI field, but is rather intrinsic to the subject matter. And if these notions are correct, then quite likely the future rigorous science of AGI will contain formal theorems echoing and improving the qualitative observations and conjectures we've made here.

If the ideas sketched here are true, then the practical consequence for AGI development is, very simply, that one shouldn't worry a lot about producing intermediary results that are compelling to skeptical observers. Just at 2/3 of a human brain may not be much use, similarly, 2/3 of an AGI system may not be much use. Lack of impressive intermediary results may not imply one is on a wrong development path; and comparison with narrow AI systems on specific tasks may be badly misleading as a gauge of incremental progress toward human-level AGI.

Hopefully it's clear that the motivation behind the line of thinking presented here is a desire to understand the nature of general intelligence and its pursuit – not a desire to avoid testing our AGI software! Really, as AGI engineers, we would love to have a sensible rigorous way to test our intermediary progress toward AGI, so as to be able to pose convincing arguments to skeptics, funding sources, potential collaborators and so forth. Our motivation here is not a desire to avoid having the intermediate progress of our efforts measured, but rather a desire to explain the frustrating (but by now rather well-established) difficulty of creating such intermediate goals for human-level AGI in a meaningful way.

If we or someone else figures out a compelling way to measure partial progress toward AGI, we will celebrate the occasion. But it seems worth seriously considering the possibility that the difficulty in finding such a measure reflects fundamental properties of general intelligence.

From a practical CogPrime perspective, we are interested in a variety of evaluation and testing methods, including the "virtual preschool" approach mentioned briefly above and more extensively in later chapters. However, our focus will be on evaluation methods that give us meaningful information about CogPrime 's progress, given our knowledge of how CogPrime works and our understanding of the underlying theory. We are unlikely to focus on the achievement of intermediate test results capable of convincing skeptics of the reality of our partial progress, because we have not yet seen any credible tests of this nature, and because we suspect the reasons for this lack may be rooted in deep properties of feasible general intelligence, such as tricky cognitive synergy.

Chapter 9
General Intelligence in the Everyday Human World

9.1 Introduction

Intelligence is not just about what happens inside a system, but also about what happens outside that system, and how the system interacts with its environment. Real-world general intelligence is about intelligence *relative to some particular class of environments*, and human-like general intelligence is about intelligence relative to the particular class of environments that humans evolved in (which in recent millennia has included environments humans have created using their intelligence). In Chapter 2, we reviewed some specific capabilities characterizing human-like general intelligence; to connect these with the general theory of general intelligence from the last few chapters, we need to explain what aspects of human-relevant *environments* correspond to these human-like intelligent *capabilities*. We begin with aspects of the environment related to communication, which turn out to tie in closely with cognitive synergy. Then we turn to physical aspects of the environment, which we suspect also connect closely with various human cognitive capabilities. In the following chapter we present a deeper, more abstract theoretical framework encompassing these ideas.

These ideas are of theoretical importance, and they're also of practical importance when one turns to the critical area of *AGI environment design*. If one is going to do anything besides release one's young AGI into the "wilds" of everyday human life, then one has to put some thought into what kind of environment it will be raised in. This may be a virtual world or it may be a robot preschool or some other kind of physical environment, but in any case some specific choices must be made about what to include. Specific choices must also be made about what kind of body to give one's AGI system – what sensors and actuators, and so forth. In Chapter 16 we will present some specific suggestions regarding choices of embodiment and environment that we find to be ideal for AGI development – virtual and robot preschools – but the material in this chapter is of more general import, beyond any such

particularities. If one has an intuitive idea of what properties of body and world human intelligence is biased for, then one can make practical choices about embodiment and environment in a principled rather than purely ad hoc or opportunistic way.

9.2 Some Broad Properties of the Everyday World That Help Structure Intelligence

The properties of the everyday world that help structure intelligence are diverse and span multiple levels of abstraction. Most of this chapter will focus on fairly concrete patterns of this nature, such as are involved in inter-agent communication and naive physics; however, it's also worth noting the potential importance of more abstract patterns distinguishing the everyday world from arbitrary mathematical environments.

The propensity to search for hierarchical patterns is one huge potential example of an abstract everyday-world property. We strongly suspect the reason that searching for hierarchical patterns works so well, in so many everyday-world contexts, lies in the particular structure of the everyday world – it's not something that would be true across all possible environments (even if one weights the space of possible environments in some clever way, say using program-length according to some standard computational model). However, this sort of assertion is of course highly "philosophical," and becomes complex to formulate and defend convincingly given the current state of science and mathematics.

Going one step further, we recall from Chapter 3 a structure called the "dual network", which consists of superposed hierarchical and heterarchical networks: basically a hierarchy in which the distance between two nodes in the hierarchy is correlated with the distance between the nodes in some metric space. Another high level property of the everyday world may be that dual network structures are prevalent. This would imply that minds biased to represent the world in terms of dual network structure are likely to be intelligent with respect to the everyday world.

In a different direction, the extreme commonality of symmetry groups in the (everyday and otherwise) physical world is another example: they occur so often that minds oriented toward recognizing patterns involving symmetry groups are likely to be intelligent with respect to the real world.

We suspect that the number of cognitively-relevant properties of the everyday world is huge ... and that the essence of everyday-world intelligence lies in the list of varyingly abstract and concrete properties, which must be embedded implicitly or explicitly in the structure of a natural or artificial intelligence for that system to have everyday-world intelligence.

Apart from these particular yet abstract properties of the everyday world, intelligence is just about "finding patterns in which actions tend to achieve

which goals in which situations" ... but, the simple meta-algorithm needed to accomplish this universally is, we suggest, only a small percentage what it takes to make a mind.

You might say that a sufficiently generally intelligent system should be able to infer the various cognitively-relevant properties of the environment from looking at data about the everyday world. We agree *in principle*, and in fact Ben Kuipers and his colleagues have done some interesting work in this direction, showing that learning algorithms can infer some basics about the structure of space and time from experience [**?**]. But we suggest that doing this really thoroughly would require a massively greater amount of processing power than an AGI that embodies and hence automatically utilizes these principles? It may be that the problem of inferring these properties is so hard as to require a wildly infeasible $AIXI^{tl}$ / Godel Machine type system.

9.3 Embodied Communication

Next we turn to the potential cognitive implications of seeking to achieve goals in an environment in which multimodal communication with other agents plays a prominent role.

Consider a community of embodied agents living in a shared world, and suppose that the agents can communicate with each other via a set of mechanisms including:

- **Linguistic communication** , in a language whose semantics is largely (not necessarily wholly) interpretable based on the mutually experienced world
- **Indicative communication** , in which e.g. one agent points to some part of the world or delimits some interval of time, and another agent is able to interpret the meaning
- **Demonstrative communication** , in which an agent carries out a set of actions in the world, and the other agent is able to imitate these actions, or instruct another agent as to how to imitate these actions
- **Depictive communication** , in which an agent creates some sort of (visual, auditory, etc.) construction to show another agent, with a goal of causing the other agent to experience phenomena similar to what they would experience upon experiencing some particular entity in the shared environment
- **Intentional communication** , in which an agent explicitly communicates to another agent what its goal is in a certain situation [1]

It is clear that ordinary everyday communication between humans possesses all these aspects.

[1] in Appendix C we recount some interesting recent results showing that mirror neurons fire in response to some cases of intentional communication as thus defined

We define the **Embodied Communication Prior** (ECP) as the probability distribution in which the probability of an entity (e.g. a goal or environment) is proportional to the difficulty of describing that entity, for a typical member of the community in question, using a particular set of communication mechanisms including the above five modes. We will sometimes refer to the prior probability of an entity under this distribution, as its "simplicity" under the distribution.

Next, to further specialize the Embodied Communication Prior, we will assume that for each of these modes of communication, there are some aspects of the world that are much more easily communicable using that mode than the other modes. For instance, in the human everyday world:

- Abstract (declarative) statements spanning large classes of situations are generally much easier to communicate linguistically
- Complex, multi-part procedures are much easier to communicate either demonstratively, or using a combination of demonstration with other modes
- Sensory or episodic data is often much easier to communicate demonstratively
- The current value of attending to some portion of the shared environment is often much easier to communicate indicatively
- Information about what goals to follow in a certain situation is often much easier to communicate intentionally, i.e. via explicitly indicating what one's own goal is

These simple observations have significant implications for the nature of the Embodied Communication Prior. For one thing they let us define multiple forms of knowledge:

- **Isolatedly declarative knowledge** is that which is much more easily communicable linguistically
- **Isolatedly procedural knowledge** is that which is much more easily communicable demonstratively
- **Isolatedly sensory knowledge** is that which is much more easily communicable depictively
- **Isolatedly attentive knowledge** is that which is much more easily communicable indicatively
- **Isolatedly intentional knowledge** is that which is much more easily communicable intentionally

This categorization of knowledge types resembles many ideas from the cognitive theory of memory [TC05], although the distinctions drawn here are a little crisper than any classification currently derivable from available neurological or psychological data.

Of course there may be much knowledge, of relevance to systems seeking intelligence according to the ECP, that does not fall into any of these

categories and constitutes "mixed knowledge." There are some very important specific subclasses of mixed knowledge. For instance, episodic knowledge (knowledge about specific real or hypothetical sets of events) will most easily be communicated via a combination of declarative, sensory and (in some cases) procedural communication. Scientific and mathematical knowledge are generally mixed knowledge, as is most everyday commonsense knowledge.

Some cases of mixed knowledge are reasonably well decomposable, in the sense that they decompose into knowledge items that individually fall into some specific knowledge type. For instance, an experimental chemistry procedure may be much better communicable procedurally, whereas an allied piece of knowledge from theoretical chemistry may be much better communicable declaratively; but in order to fully communicate either the experimental procedure or the abstract piece of knowledge, one may ultimately need to communicate both aspects.

Also, even when the best way to communicate something is mixed-mode, it may be possible to identify one mode that poses the most important part of the communication. An example would be a chemistry experiment that is best communicated via a practical demonstration together with a running narrative. It may be that the demonstration without the narrative would be vastly more valuable than the narrative without the demonstration. To cover such cases we may make less restrictive definitions such as

- **Interactively declarative knowledge** is that which is much more easily communicable in a manner dominated by linguistic communication

and so forth. We call these "interactive knowledge categories," by contrast to the "isolated knowledge categories" introduced earlier.

9.3.0.1 Naturalness of Knowledge Categories

Next we introduce an assumption we call NKC, for Naturalness of Knowledge Categories. The NKC assumption states that the knowledge in each of the above isolated and interactive communication-modality-focused categories forms a "natural category," in the sense that for each of these categories, there are many different properties shared by a large percentage of the knowledge in the category, but not by a large percentage of the knowledge in the other categories. This means that, for instance, procedural knowledge systematically (and statistically) has different characteristics than the other kinds of knowledge.

The NKC assumption seems commonsensically to hold true for human everyday knowledge, and it has fairly dramatic implications for general intelligence. Suppose we conceive general intelligence as the ability to achieve goals in the environment shared by the communicating agents underlying the Embodied Communication Prior. Then, NKC suggests that the best way to

achieve general intelligence according to the Embodied Communication Prior is going to involve

- specialized methods for handling declarative, procedural, sensory and attentional knowledge (due to the naturalness of the isolated knowledge categories)
- specialized methods for handling interactions between different types of knowledge, including methods focused on the case where one type of knowledge is primary and the others are supporting (the latter due to the naturalness of the interactive knowledge categories)

9.3.0.2 Cognitive Completeness

Suppose we conceive an AI system as consisting of a set of learning capabilities, each one characterized by three features:

- One or more **knowledge types** that it is competent to deal with, in the sense of the two key learning problems mentioned above
- At least one **learning type** : either analysis, or synthesis, or both
- At least one **interaction type** , for each (knowledge type, learning type) pair it handles: "isolated" (meaning it deals mainly with that knowledge type in isolation), or "interactive" (meaning it focuses on that knowledge type but in a way that explicitly incorporates other knowledge types into its process), or "fully mixed" (meaning that when it deals with the knowledge type in question, no particular knowledge type tends to dominate the learning process).

Then, intuitively, it seems to follow from the ECP with NKC that systems with high efficient general intelligence should have the following properties, which collectively we'll call **cognitive completeness**:

- For each (knowledge type, learning type, interaction type) triple, there should be a learning capability corresponding to that triple.
- Furthermore the capabilities corresponding to different (knowledge type, interaction type) pairs should have distinct characteristics (since according to the NKC the isolated knowledge corresponding to a knowledge type is a natural category, as is the dominant knowledge corresponding to a knowledge type)
- For each (knowledge type, learning type) pair (K,L), and each other knowledge type K1 distinct from K, there should be a distinctive capability with interaction type "interactive" and dealing with knowledge that is interactively K but also includes aspects of K1

Furthermore, it seems intuitively sensible that according to the ECP with NKC, if the capabilities mentioned in the above points are reasonably able,

then the system possessing the capabilities will display general intelligence relative to the ECP. Thus we arrive at the hypothesis that

Under the assumption of the Embodied Communication Prior (with the Natural Knowledge Categories assumption), the property above called "cognitive completeness" is necessary and sufficient for efficient general intelligence at the level of an inteligent adult human (e.g. at the Piagetan formal level [Pia53].

Of course, the above considerations are very far from a rigorous mathematical proof (or even precise formulation) of this hypothesis. But we are presenting this here as a conceptual hypothesis, in order to qualitatively guide our practical AGI R&D and also to motivate further, more rigorous theoretical work.

9.3.1 Generalizing the Embodied Communication Prior

One interesting direction for further research would be to broaden the scope of the inquiry, in a manner suggested above: instead of just looking at the ECP, look at simplicity measures in general, and attack the question of how a mind must be structured in order to display efficient general intelligence relative to a specified simplicity measure. This problem seems unapproachable in general, but some special cases may be more tractable.

For instance, suppose one has

- a simplicity measure that (like the ECP) is approximately decomposable into a set of fairly distinct components, plus their interactions
- an assumption similar to NKC, which states that the entities displaying simplicity according to each of the distinct components, are roughly clustered together in entity-space

Then one should be able to say that, to achieve efficient general intelligence relative to this decomposable simplicity measure, a system should have distinct capabilities corresponding to each of the components of the simplicity measure interactions between these capabilities, corresponding to the interaction terms in the simplicity measure

With copious additional work, these simple observations could potentially serve as the seed for a novel sort of theory of general intelligence - a theory of how the structure of a system depends on the structure of the simplicity measure with which it achieves efficient general intelligence. Cognitive Synergy Theory would then emerge as a special case of this more abstract theory.

9.4 Naive Physics

Multimodal communication is an important aspect of the environment for which human intelligence evolved – but not the only one. It seems likely that our human intelligence is also closely adapted to various aspects of our physical environment – a matter that is worth carefully attending as we design environments for our robotically or virtually embodied AGI systems to operate in.

One interesting guide to the most cognitively relevant aspects of human environments is the subfield of AI known as "naive physics" [**?**] – a term that refers to the theories about the physical world that human beings implicitly develop and utilize during their lives. For instance, when you figure out that you need to pressure the knife slightly harder when spreading peanut butter rather than jelly, you're not making this judgment using Newtonian physics or the Navier-Stokes equations of fluid dynamics; you're using heuristic patterns that you figured out through experience. Maybe you figured out these patterns through experience spreading peanut butter and jelly in particular. Or maybe you figured these heuristic patterns out before you ever tried to spread peanut butter or jelly specifically, via just touching peanut butter and jelly to see what they feel like, and then carrying out inference based on your experience manipulating similar tools in the context of similar substances.

Other examples of similar "naive physics" patterns are easy to come by, e.g.

1. What goes up must come down.
2. A dropped object falls straight down.
3. A vacuum sucks things towards it.
4. Centrifugal force throws rotating things outwards.
5. An object is either at rest or moving, in an absolute sense.
6. Two events are simultaneous or they are not.
7. When running downhill, one must lift one's knees up high
8. When looking at something that you just barely can't discern accurately, squint

Attempts to axiomatically formulate naive physics have historically come up short, and we doubt this is a promising direction for AGI. However, we do think the naive physics literature does a good job of identifying the various phenomena that the human mind's naive physics deals with. So, from the point of view of AGI environment design, naive physics is a useful source of requirements. Ideally, we would like an AGI's environment to support all the fundamental phenomena that naive physics deals with.

We now describe some key aspects of naive physics in a more systematic manner. Naive physics has many different formulations; in this section we draw heavily on [SC94], who divide naive physics phenomena into 5 categories. Here we review these categories and identify a number of important

things that humanlike intelligent agents must be able to do relative to each of them.

9.4.1 Objects, Natural Units and Natural Kinds

One key aspect of naive physics involves recognition of various aspects of objects, such as:

1. Recognition of objects amidst noisy perceptual data
2. Recognition of surfaces and interiors of objects
3. Recognition of objects as manipulable units
4. Recognition of objects as potential subjects of fragmentation (splitting, cutting) and of unification (gluing, bonding)
5. Recognition of the agent's body as an object, and as parts of the agent's body as objects
6. Division of universe of perceived objects into "natural kinds", each containing typical and atypical instances

9.4.2 Events, Processes and Causality

Specific aspects of naive physics related to temporality and causality are:

1. Distinguishing roughly-subjectively-instantaneous events from extended processes
2. Identifying beginnings, endings and crossings of processes.
3. Identifying and distinguishing internal and external changes
4. Identifying and distinguishing internal and external changes relative to one's own body
5. Interrelating body-changes with changes in external entities

Notably, these aspects of naive physics involve a different processes occurring on a variety of different time scales, intersecting in complex patterns, and involving processes inside the agent's body, outside the agent's body, and crossing the boundary of the agent's body.

9.4.3 Stuffs, States of Matter, Qualities

Regarding the various states of matter, some important aspects of naive physics are:

1. Perceiving gaps between objects: holes, media, illusions like rainbows, mirages and holograms
2. Distinguishing the manners in which different sorts of entities (e.g. smells, sounds, light) fill space
3. Distinguishing properties such as smoothness, roughness, graininess, stickiness, runniness, etc.
4. Distinguishing degrees of elasticity and fragility
5. Assessing separability of aggregates

9.4.4 Surfaces, Limits, Boundaries, Media

Gibson [Gib77, Gib79] has argued that naive physics is not mainly about objects but rather mainly about surfaces. Surfaces have a variety of aspects and relationships that are important for naive physics, such as:

1. Perceiving and reasoning about surfaces as two-sided or one-sided interfaces
2. Inference of the various ecological laws of surfaces
3. Perception of various media in the world as separated by surfaces
4. Recognition of the textures of surfaces
5. Recognition of medium/surface layout relationships such as: ground, open environment, enclosure, detached object, attached object, hollow object, place, sheet, fissure, stick, fibre, dihedral, etc.

Fig. 9.1 One of Sloman's example test domains for real-world inference. Left: a number of pins and a rubber band to be stretched around them. Right: use of the pins and rubber band to make a letter T.

As a concrete, evocative "toy" example of naive everyday knowledge about surfaces and boundaries, consider Sloman's [Slo08] example scenario, depicted in Figure 9.1 and drawn largely from [SS74] (see also related discussion in [?], in which "A child can be given one or more rubber bands and a pile of

pins, and asked to use the pins to hold the band in place to form a particular shape.... For example, things to be learnt could include":

1. There is an area inside the band and an area outside the band
2. The possible effects of moving a pin that is inside the band towards or further away from other pins inside the band. (The effects can depend on whether the band is already stretched.)
3. The possible effects of moving a pin that is outside the band towards or further away from other pins inside the band.
4. The possible effects of adding a new pin, inside or outside the band, with or without pushing the band sideways with the pin first.
5. The possible effects of removing a pin, from a position inside or outside the band.
6. Patterns of motion/change that can occur and how they affect local and global shape (e.g. introducing a concavity or convexity, introducing or removing symmetry, increasing or decreasing the area enclosed).
7. The possibility of causing the band to cross over itself. (NB: Is an odd number of crosses possible?)
8. How adding a second, or third band can enrich the space of structures, processes and effects of processes.

9.4.5 What Kind of Physics Is Needed to Foster Human-like Intelligence?

We stated above that we would like an AGI's environment to support all the fundamental phenomena that naive physics deals with; and we have now reviewed a number of these specific phenomena. But it's not entirely clear what the "fundamental" aspects underlying these phenomena' are. One important question in the environment-design context is how close an AGI environment needs to stick to the particulars of real-world naive physics. Is it important that a young AGI can play with the specific differences between spreading peanut butter versus jelly? Or is it enough that it can play with spreading and smearing various substances of different consistencies? How close does the analogy between an AGI environment's naive physics and real-world naive physics need to be? This is a question to which we have no scientific answer at present. Our own working hypothesis is that the analogy does not need to be extremely close, and with this in mind in Chapter 16 we propose a virtual environment BlocksNBeadsWorld that encompasses all the basic conceptual phenomena of real-world naive physics, but does not attempt to emulate their details.

Framed in terms of human psychology rather than environment design, the question becomes: *At what level of detail must one model the physical world to understand the ways in which human intelligence has adapted to*

the physical world?. Our suspicion, which underlies our BlocksNBeadsWorld design, is that it's approximately enough to have

- Newtonian physics, or some close approximation
- Matter in multiple phases and forms vaguely similar to the ones we see in the real world: solid, liquid, gas, paste, goo, etc.
- Ability to transform some instances of matter from one form to another
- Ability to flexibly manipulate matter in various forms with various solid tools
- Ability to combine instances of matter into new ones in a fairly rich way: e.g. glue or tie solids togethermix liquids together, etc.
- Ability to position instances of matter with respect to each other in a rich way: e.g. put liquid in a solid cavity, cover something with a lid or a piece of fabric, etc.

It seems to us that if the above are present in an environment, then an AGI seeking to achieve appropriate goals in that environment will be likely to form an appropriate "human-like physical-world intuition." We doubt that the specifics of the naive physics of different forms of matter are critical to human-like intelligence. But, we suspect that a great amount of unconscious human metaphorical thinking is conditioned on the fact that humans evolved around matter that takes a variety of forms, can be changed from one form to another, and can be fairly easily arranged and composited to form new instances from prior ones. Without many diverse instances of matter transformation, arrangement and composition in its experience, an AGI is unlikely to form an internal "metaphor-base" even vaguely similar to the human one – so that, even if it's highly intelligent, its thinking will be radically non-human-like in character.

Naturally this is all somewhat speculative and must be explored via experimentation. Maybe an elaborate blocks-world with only solid objects will be sufficient to create human-level, roughly human-like AGI with rich spatiotemporal and manipulative intuition. Or maybe human intelligence is more closely adapted to the specifics of our physical world – with water and dirt and plants and hair and so forth – than we currently realize. One thing that *is* very clear is that, as we proceed with embodying, situating and educating our AGI systems, we need to pay careful attention to the way their intelligence is conditioned by their environment.

9.5 Folk Psychology

Related to naive physics is the notion of "naive psychology" or "folk psychology" [Rav04], which includes for instance the following aspects:

1. Mental simulation of other agents

2. Mental theory regarding other agents
3. Attribution of beliefs, desires and intentions (BDI) to other agents via theory or simulation
4. Recognition of emotions in other agents via their physical embodiment
5. Recognition of desires and intentions in other agents via their physical embodiment
6. Analogical and contextual inferences between self and other, regarding BDI and other aspects
7. Attribute causes and meanings to other agents behaviors
8. Anthropomorphize non-human, including inanimate objects

The main special requirement placed on an AGI's embodiment by the above aspects pertains to the ability of agents to express their emotions and intentions to each other. Humans do this via facial expressions and gestures.

9.5.1 Motivation, Requiredness, Value

Relatedly to folk psychology, Gestalt [**?**] and ecological [Gib77, Gib79] psychology suggest that humans perceive the world substantially in terms of the affordances it provides them for goal-directed action. This suggests that, to support human-like intelligence, an AGI must be capable of:

1. Perception of entities in the world as differentially associated with goal-relevant value
2. Perception of entities in the world in terms of the potential actions they afford the agent, or other agents

The key point is that entities in the world need to provide a wide variety of ways for agents to interact with them, enabling richly complex perception of affordances.

9.6 Body and Mind

The above discussion has focused on the world external to the body of the AGI agent embodied and embedded in the world, but the issue of the AGIs body also merits consideration. There seems little doubt that a human's intelligence is highly conditioned by the particularities human body.

Here the requirements seem fairly simple: while surely not strictly necessary, it would certainly be *preferable* to provide an AGI with fairly rich analogues of the human senses of touch, sight, sound, kinesthesia, taste and smell. Each of these senses provides different sorts of cognitive stimulation to the human mind; and while similar cognitive stimulation could doubtless

be achieved without analogous senses, the provision of such seems the most straightforward approach. It's hard to know how much of human intelligence is specifically biased to the sorts of outputs provided by human senses.

As vision already is accorded such a prominent role in the AI and cognitive science literature – and is discussed in moderate depth in Chapter 26 of Part 2, we won't take time elaborating on the importance of vision processing for humanlike cognition. The key thing an AGI requires to support humanlike "visual intelligence" is an environment containing a sufficiently robust collection of materials that object and event recognition and identification become interesting problems.

Audition is cognitively valuable for many reasons, one of which is that it gives a very rich and precise method of sensing the world that is different from vision. The fact that humans can display normal intelligence while totally blind or totally deaf is an indication that, in a sense, vision and audition are redundant for understanding the everyday world. However, it may be important that the brain has evolved to account for both of these senses, because this forced it to account for the presence of two very rich and precise methods of sensing the world – which may have forced it to develop more abstract representation mechanisms than would have been necessary with only one such method.

Touch is a sense that is, in our view, generally badly underappreciated within the AI community. In particular the cognitive robotics community seems to worry too little about the terribly impoverished sense of touch possessed by most current robots (though fortunately there are recent technologies that may help improve robots in this regard; see e.g. [Nan08]). Touch is how the human infant learns to distinguish self from other, and in this way it is the most essential sense for the establishment of an internal self-model. Touching others' bodies is a key method for developing a sense of the emotional reality and responsiveness of others, and is hence key to the development of theory of mind and social understanding in humans. For this reason, among others, human children lacking sufficient tactile stimulation will generally wind up badly impaired in multiple ways. A good-quality embodiment should supply an AI agent with a body that possesses skin, which has varying levels of sensitivity on different parts of the skin (so that it can effectively distinguish between reality and its perception thereof in a tactile context); and also varying types of touch sensors (e.g. temperature versus friction), so that it experiences textures as multidimensional entities.

Related to touch, kinesthesia refers to direct sensation of phenomena happening inside the body. Rarely mentioned in AI, this sense seems quite critical to cognition, as it underpins many of the analogies between self and other that guide cognition. Again, it's not important that an AGI's virtual body have the same internal body parts as a human body. But it seems valuable to have the AGI's virtual body display some vaguely human-body-like properties, such as feeling internal strain of various sorts after getting exercise,

feeling discomfort in certain places when running out of energy, feeling internally different when satisfied versus unsatisfied, etc.

Next, taste is a cognitively interesting sense in that it involves the interplay between the internal and external world; it involves the evaluation of which entities from the external world are worthy of placing inside the body. And smell is cognitively interesting in large part because of its relationship with taste. A smell is, among other things, a long-distance indicator of what a certain entity might taste like. So, the combination of taste and smell provides means for conceptualizing relationships between self, world and distance. However, our sense is that these are cognitive niceties rather than necessities, so that the provision of an AGI with humanlike taste and smell is almost surely cognitively irrelevant.

9.7 The Extended Mind and Body

Finally, Hutchins [Hut95], Logan [Log07] and others have promoted a view of human intelligence that views the human mind as extended beyond the individual body, incorporating social interactions and also interactions with inanimate objects, such as tools, plants and animals. This leads to a number of requirements for a humanlike AGI's environment:

1. The ability to create a variety of different tools for interacting with various aspects of the world in various different ways, including tools for making tools and ultimately machinery
2. The existence of other mobile, virtual life-forms in the world, including simpler and less intelligent ones, and ones that interact with each other and with the AGI
3. The existence of organic growing structures in the world, with which the AGI can interact in various ways, including halting their growth or modifying their growth pattern

How necessary these requirements are is hard to say – but it *is* clear that these things have played a major role in the evolution of human intelligence.

9.8 Conclusion

Happily, this long and diverse chapter supports a simple, albeit tentative conclusion. Our suggestion is that, if an AGI is

- placed in an environment capable of roughly supporting multimodal communication and vaguely (but not necessarily precisely) real-world-ish naive physics

- surrounded with other intelligent agents of varying levels of complexity, and other complex, dynamic structures to interface with
- given a body that can perceive this environment through some forms of sight, sound and touch; and perceive itself via some form of kinesthesia
- given a motivational system that encourages it to make rich use of these aspects of its environment

then the AGI is likely to have an experience-base reinforcing the key inductive biases provided by the everyday world for the guidance of humanlike intelligence.

Chapter 10
A Mind-World Correspondence Principle

10.1 Introduction

Real-world minds are always adapted to certain classes of environments and goals. The ideas of the previous chapter, regarding the connection between a human-like intelligence's internals and its environment, result from exploring the implications of this adaptation in the context of the cognitive synergy concept. In this chapter we explore the mind-world connection in a broader and more abstract way – making a more ambitious attempt to move toward a "general theory of general intelligence."

One basic premise here, as in the preceding chapters is: Even a system of vast general intelligence, subject to real-world space and time constraints, will necessarily be more efficient at some kinds of learning than others. Thus, one approach to formulating a general theory of general intelligence is to look at the relationship between minds and worlds Ð where a ÒworldÓ is conceived as an environment and a set of goals defined in terms of that environment.

In this spirit, we here formulate a broad principle binding together worlds and the minds that are intelligent in thee worlds. The ideas of the previous chapter constitute specific, concrete instantiations of this general principle. A careful statement of the principle requires introduction of a number of technical concepts, and will be given later on in the chapter. A crude, informal version of the principle would be:

MIND-WORLD CORRESPONDENCE-PRINCIPLE

. For a mind to work intelligently toward certain goals in a certain world, there should be a nice mapping from goal-directed sequences of world-states into sequences of mind-states, where ÒniceÓ means that a world-state-sequence W composed of two parts W_1 and W_2, gets mapped into a mind-state-sequence M composed of two corresponding parts M_1 and M_2.

What's nice about this principle is that it relates the decomposition of the world into parts, to the decomposition of the mind into parts.

10.2 What Might a General Theory of General Intelligence Look Like?

It's not clear, at this point, what a real Ògeneral theory of general intelligenceÓ would look like – but one tantalizing possibility is that it might confront the two questions:

- How does one design a world to foster the development of a certain sort of mind?
- How does one design a mind to match the particular challenges posed by a certain sort of world?

One way to achieve this would be to create a theory that, given a description of an environment and some associated goals, would output a description of the structure and dynamics that a system should possess to be intelligent in that environment relative to those goals, using limited computational resources.

Such a theory would serve a different purpose from the mathematical theory of "universal intelligence" developed by Marcus Hutter [Hut05] and others. For all its beauty and theoretical power, that approach currently it gives useful conclusions only about general intelligences with infinite or infeasibly massive computational resources. On the other hand, the approach suggested here is aimed toward creation of a theory of real-world general intelligences utilizing realistic amounts of computational power, but still possessing general intelligence comparable to human beings or greater.

This reflects a vision of intelligence as largely concerned with adaptation to particular classes of environments and goals. This may seem contradictory to the notion of ÒgeneralÓ intelligence, but I think it actually embodies a realistic understanding of general intelligence. Maximally general intelligence is not pragmatically feasible; it could only be achieved using infinite computational resources [Hut05]. Real-world systems are inevitably limited in the intelligence they can display in any real situation, because real situations involve finite resources, including finite amounts of time. One may say that, in principle, a certain system could solve any problem given enough resources and time Ð but, even when this is true, it's not necessarily the most interesting way to look at the system's intelligence. It may be more important to look at what a system can do given the resources at its disposal in reality. And this perspective leads one to ask questions like the ones posed above: which bounded-resources systems are well-disposed to display intelligence in which classes of situations?

As noted in Chapter 7 above, one can assess the generality of a system's intelligence via looking at the entropy of the class of situations across which it displays a high level of intelligence (where "high" is measured relative to its total level of intelligence across all situations). A system with a high generality of intelligence will tend to be roughly equally intelligent across a wide variety of situations; whereas a system with lower generality of intelligence will tend to be much more intelligent in a small subclass of situations, than in any other. The definitions given above embody this notion in a formal and quantitative way.

If one wishes to create a general theory of general intelligence according to this sort of perspective, the main question then becomes how to represent goals/environments and systems in such a way as to render transparent the natural correspondence between the specifics of the former and the latter, in the context of resource-bounded intelligence. This is the business of the next section.

10.3 Steps Toward A (Formal) General Theory of General Intelligence

Now begins the formalism. At this stage of development of the theory proposed in this chapter, mathematics is used mainly as a device to ensure clarity of expression. However, once the theory is further developed, it may possibly become useful for purposes of calculation as well.

Suppose one has any system S (which could be an AI system, or a human, or an environment that a human or AI is interacting with, or the combination of an environment and a human or AI's body, etc.). One may then construct an uncertain transition graph associated with that system S, in the following way:

- The nodes of the graph represent fuzzy sets of states of system S (I'll call these Òstate-setsÓ from here on, leaving the fuzziness implicit)
- The (directed) links of the graph represent probabilistically weighted transitions between state-sets

Specifically, the weight of the link from A to B should be defined as

$$P(o(S, A, t(T))|o(S, B, T))$$

where

$$o(S, A, T)$$

denotes the presence of the system S in the state-set A during time-distribution T, and $t()$ is a temporal succession function defined so that $t(T)$ refers to a time-distribution conceived as ÒafterÓ T. A time-distribution is

a probability distribution over time-points. The interaction of fuzziness and probability here is fairly straightforward and may be handled in the manner of PLN, as outlined in subsequent chapters. Note that the definition of link weights is dependent on the specific implementation of the temporal succession function, which includes an implicit time-scale.

Suppose one has a transition graph corresponding to an environment; then a goal relative to that environment may be defined as a particular node in the transition graph. The goals of a particular system acting in that environment may then be conceived as one or more nodes in the transition graph. The system's situation in the environment at any point in time may also be associated with one or more nodes in the transition graph; then, the system's movement toward goal-achievement may be associated with paths through the environment's transition graph leading from its current state to goal states.

It may be useful for some purposes to filter the uncertain transition graph into a crisp transition graph by placing a threshold on the link weights, and removing links with weights below the threshold.

The next concept to introduce is the world-mind transfer function, which maps world (environment) state-sets into organism (e.g. AI system) state-sets in a specific way. Given a world state-set W, the world-mind transfer function M maps W into various organism state-sets with various probabilities, so that we may say: $M(W)$ is the probability distribution of state-sets the organism tends to be in, when its environment is in state-set W. (Recall also that state-sets are fuzzy.)

Now one may look at the spaces of world-paths and mind-paths. A world-path is a path through the world's transition graph, and a mind-path is a path through the organism's transition graph. Given two world-paths P and Q, it's obvious how to define the composition $P*Q$ Ð one follows P and then, after that, follows Q, thus obtaining a longer path. Similarly for mind-paths.

In category theory terms, we are constructing the free category associated with the graph: the objects of the category are the nodes, and the morphisms of the category are the paths. And category theory is the right way to be thinking here Ð we want to be thinking about the relationship between the world category and the mind category.

The world-mind transfer function can be interpreted as a mapping from paths to subgraphs: Given a world-path, it produces a set of mind state-sets, which have a number of links between them. One can then define a world-mind path transfer function $M(P)$ via taking the mind-graph $M(nodes(P))$, and looking at the highest-weight path spanning $M(nodes(P))$. (Here *nodes*? obviously means the set of nodes of the path P.)

A functor F between the world category and the mind category is a mapping that preserves object identities and so that

$$F(P * Q) = F(P) * F(Q)$$

We may also introduce the notion of an approximate functor, meaning a mapping F so that the average of

$$d(F(P * Q), F(P) * F(Q))$$

is small.

One can introduce a prior distribution into the average here. This could be the Levin universal distribution or some variant (the Levin distribution assigns higher probability to computationally simpler entities). Or it could be something more purpose specific: for example, one can give a higher weight to paths leading toward a certain set of nodes (e.g. goal nodes). Or one can use a distribution that weights based on a combination of simplicity and directedness toward a certain set of nodes. The latter seems most interesting, and I will define a goal-weighted approximate functor as an approximate functor, defined with averaging relative to a distribution that balances simplicity with directedness toward a certain set of goal nodes.

The move to approximate functors is simple conceptually, but mathematically it's a fairly big step, because it requires us to introduce a geometric structure on our categories. But there are plenty of natural metrics defined on paths in graphs (weighted or not), so there's no real problem here.

10.4 The Mind-World Correspondence Principle

Now we finally have the formalism set up to make a non-trivial statement about the relationship between minds and worlds. Namely, the hypothesis that:

MIND-WORLD CORRESPONDENCE PRINCIPLE

: For an organism with a reasonably high level of intelligence in a certain world, relative to a certain set of goals, the mind-world path transfer function is a goal-weighted approximate functor

That is, a little more loosely: the hypothesis is that, for intelligence to occur, there has to be a natural correspondence between the transition-sequences of world-states and the corresponding transition-sequences of mind-states, at least in the cases of transition-sequences leading to relevant goals.

We suspect that a variant of the above proposition can be formally proved, using the definition of general intelligence presented in Chapter 7. The proof of a theorem corresponding to the above would certainly constitute an interesting start toward a general formal theory of general intelligence. Note that proving anything of this nature would require some attention to the time-scale-dependence of the link weights in the transition graphs involved.

A formally proved variant of the above proposition would be in short, a "MIND-WORLD CORRESPONDENCE THEOREM."

Recall that at the start of the chapter, we expressed the same idea as:

MIND-WORLD CORRESPONDENCE-PRINCIPLE

: For a mind to work intelligently toward certain goals in a certain world, there should be a nice mapping from goal-directed sequences of world-states into sequences of mind-states, where ÒniceÓ means that a world-state-sequence W composed of two parts W_1 and W_2, gets mapped into a mind-state-sequence M composed of two corresponding parts M_1 and M_2.

That is a reasonable gloss of the principle, but it's clunkier and less accurate, than the statement in terms of functors and path transfer functions, because it tries to use only common-language vocabulary, which doesn't really contain all the needed concepts.

10.5 How Might the Mind-World Correspondence Principle Be Useful?

Suppose one believes the Mind-World Correspondence Principle as laid out above Ð so what?

Our hope, obviously, is that the principle could be useful in actually figuring out how to architect intelligent systems biased toward particular sorts of environment. And of course, this is said with the understanding that any finite intelligence must be biased toward some sorts of environment.

Relatedly, given a specific AGI design (such as CogPrime), one could use the principle to figure out which environments it would be best suited for. Or one could figure out how to adjust the particulars of the design, to maximize the system's intelligence in the environments of interest.

One next step in developing this network of ideas, aside from (and potentially building on) full formalization of the principle, would be an exploration of real-world environments in terms of transition graphs. What properties do the transition graphs induced from the real world have?

One such property, we suggest, is successive refinement. Often the path toward a goal involves first gaining an approximate understanding of a situation, then a slightly more accurate understanding, and so forth Ð until finally one has achieved a detailed enough understanding to actually achieve the goal. This would be represented by a world-path whose nodes are state-sets involving the gathering of progressively more detailed information.

Via pursuing to the mind-world correspondence property in this context, I believe we will find that world-paths reflecting successive refinement correspond to mind-paths embodying successive refinement. This will be found

to relate to the hierarchical structures found so frequently in both the physical world and the human mind-brain. Hierarchical structures allow many relevant goals to be approached via successive refinement, which I believe is the ultimate reason why hierarchical structures are so common in the human mind-brain.

Another next step would be exploring what mind-world correspondence means for the structure and dynamics of a limited-resources intelligence. If an organism Ohas limited resources and, to be intelligent, needs to make

$$P(o(O, M(A), t(T))|o(O, M(B), T))$$

high for particular world state-sets A and B, then what's the organism's best approach? Arguably, it should represent $M(A)$ and $M(B)$ internally in such a way that very little computational effort is required for it to transition between $M(A)$ and $M(B)$. For instance, this could be done by coding its knowledge in such a way that $M(A)$ and $M(B)$ share many common bits; or it could be done in other more complicated ways.

If, for instance, A is a subset of B, then it may prove beneficial for the organism to represent $M(A)$ physically as a subset of its representation of $M(B)$.

Pursuing this line of thinking, one could likely derive specific properties of an intelligent organism's internal information-flow, from properties of the environment and goals with respect to which it's supposed to be intelligent.

This would allow us to achieve the holy grail of intelligence theory as I understand it: given a description of an environment and goals, to be able to derive an architectural description for an organism that will display a high level of intelligence relative to those goals, given limited computational resources.

While this "holy grail" is obviously a far way off, what we've tried to do here is outline a clear mathematical and conceptual direction for moving toward it.

10.6 Conclusion

The Mind-World Correspondence Principle presented here – if in the vicinity of correctness – constitutes a non-trivial step toward fleshing out the concept of a general theory of general intelligence. But obviously the theory is still rather abstract, and also not completely rigorous. There's a lot more work to be done.

The Mind-World Correspondence Principle as articulated above is not quite a formal mathematical statement. It would take a little work to put in all the needed quantifiers to formulate it as one, and it's not clear the best way to do so Ð the details would perhaps become clear in the course

of trying to prove a version of it rigorously. One could interpret the ideas presented in this chapter as a philosophical theory that hopes to be turned into a mathematical theory and to play a key role in a scientific theory.

For the time being, the main role to be served by these ideas is qualitative: to help us think about concrete AGI designs like CogPrime in a sensible way. It's important to understand what the goal of a real-world AGI system needs to be: to achieve the ability to broadly learn and generalize, yes, but not with infinite capability É rather with biases and patterns that are implicitly and/or explicitly tuned to certain broad classes of goals and environments. The Mind-World Correspondence Principle tells us something about what this "tuning" should involve – namely, making a system possessing mind-state sequences that correspond meaningfully to world-state sequences. CogPrime 's overall design and particular cognitive processes are reasonably well interpreted as an attempt to achieve this for everyday human goals and environments.

One way of extending these theoretical ideas into a more rigorous theory is explored in Appendix B. The key ideas involved there are: modeling multiple memory types as mathematical categories (with functors mapping between them), modeling memory items as probability distributions, and measuring distance between memory items using two metrics, one based on algorithmic information theory and one on classical information geometry. Building on these ideas, core hypotheses are then presented:

- a **syntax-semantics correlation** principle, stating that in a successful AGI system, these two metrics should be roughly correlated
- a **cognitive geometrodynamics** principle, stating that on the whole intelligent minds tend to follow geodesics (shortest paths) in mindspace, according to various appropriately defined metrics (e.g. the metric measuring the distance between two entities in terms of the length and/or runtime of the shortest programs computing one from the other).
- a **cognitive synergy** principle, stating that shorter paths may be found through the composite mindspace formed by considering multiple memory types together, than by following the geodesics in the mindspaces corresponding to individual memory types.

The material is relegated to an appendix because it is so speculative, and it's not yet clear whether it will really be useful in advancing or interpreting CogPrime or other AGI systems (unlike the material from the present chapter, which has at least been useful in interpreting and tweaking the CogPrime design, even though it can't be claimed that CogPrime was derived directly from these theoretical ideas). However, this sort of speculative exploration is, in our view, exactly the sort of thing that's needed as a first phase in transitioning the ideas of the present chapter into a more powerful and directly actionable theory.

Section III
Cognitive and Ethical Development

Chapter 11
Stages of Cognitive Development

Co-authored with Stephan Vladimir Bugaj

11.1 Introduction

Creating AGI, we have said, is not only about having the right structural and dynamical possibilities implemented in the initial version of one's system – but also about the environment and embodiment that one's system is associated with, and the match between the system's internals and these externals. Another key aspect is the long-term time-course of the system's evolution over time, both in its internals and its external interaction – i.e., what is known as *development*.

Development is a critical topic in our approach to AGI because we believe that much of what constitutes human-level, human-like intelligence *emerges* in an intelligent system due to its engagement with its environment and its environment-coupled self-organization. So, it's not to be expected that the initial version of an AGI system is going to display impressive feats of intelligence, even if the engineering is totally done right. A good analogy is the apparent unintelligence of a human baby. Yes, scientists have discovered that human babies are capable of interesting and significant intelligence – but one has to hunt to find it ... at first observation, babies are rather idiotic and simple-minded creatures: much less intelligent-appearing than lizards or fish, maybe even less than cockroaches....

If the goal of an AGI project is to create an AGI system that can progressively develop advanced intelligence through learning in an environment richly populated with other agents and various inanimate stimuli and interactive entities – then an understanding of the nature of cognitive development becomes extremely important to that project.

Unfortunately, contemporary cognitive science contains essentially no theory of "abstract developmental psychology" which can conveniently be applied to understand developing AIs. There is of course an extensive science of **human** developmental psychology, and so it is a natural research program to take the chief ideas from the former and inasmuch as possible port them to

the AGI domain. This is not an entirely simple matter both because of the differences between humans and AI's and because of the unsettled nature of contemporary developmental psychology theory. But it's a job that must (and will) be done, and the ideas in this chapter may contribute toward this effort.

We will begin here with Piaget's well-known theory of human cognitive development, presenting it in a general systems theory context, then introducing some modifications and extensions and discussing some other relevant work.

11.2 Piagetan Stages in the Context of a General Systems Theory of Development

Our review of AGI architectures in Chapter 4 focused heavily on the concept of **symbolism**, and the different ways in which different classes of cognitive architecture handle symbol representation and manipulation. We also feel that symbolism is critical to the notion of AGI development – and even more broadly, to the systems theory of development in general.

As a broad conceptual perspective on development, we suggest that one may view the development of a complex information processing system, embedded in an environment, in terms of the stages:

- **automatic**: the system interacts with the environment by "instinct", according to its innate programming
- **adaptive**: the system internally adapts to the environment, then interacting with the environment in a more appropriate way
- **symbolic**: the system creates internal symbolic representations of itself and the environment, which in the case of a complex, appropriately structured environment, allows it to interact with the environment more intelligently
- **reflexive**: the system creates internal symbolic representations of its own internal symbolic representations, thus achieving an even higher degree of intelligence

Sketched so broadly these are not precisely defined categories but rather heuristic, intuitive categories. Formalizing them would be possible but would lead us too far astray here.

One can interpret these stages in a variety of different contexts. Here our focus is the cognitive development of humans and human-like AGI systems, but in Table 11.1 we present them in a slightly more general context, using two examples: the Piagetan example of the human (or humanlike) mind as it develops from infancy to maturity; and also the example of the "origin of life" and the development of life from proto-life up into its modern form. In any event, we allude to this more general perspective on development here mainly

to indicate our view that the Piagetan perspective is not something ad hoc and arbitrary, but rather can plausibly be seen as a specific manifestation of more fundamental principles of complex systems development.

Stage	**General Description**	**Cognitive Development**	**Origin of Life**
Automatic	System-environment information exchange controlled mainly by innate system structures or environment	Piagetan infantile stage	Self-organizing proto-life system, e.g. Oparin [Opa52] water droplet, or Cairns-Smith [**?**] clay-based protolife
Adaptive	System-environment info exchange heavily guided by adaptively internally-created system structures	Piagetan "concrete operational" stage: systematic internal world-model guides world-exploration	Simple autopoietic system, e.g. Oparin water droplet w/ basic metabolism
Symbolic	Internal symbolic representation of information exchange process	Piagetan formal stage: explicit logical/experimental learning about how to cognize in various contexts	Genetic code: internal entities that "stand for" aspects of organism and environment, thus enabling complex epigenesis
Reflexive	Thoroughgoing self-modification based on this symbolic representation	Piagetan post-formal stage: purposive self-modification of basic mental processes	Genes+memes: genetic code-patterns guide their own modification via influencing culture

Table 11.1 General Systems Theory of Development: Parallels Between Development of Mind and Origin of Life

11.3 Piaget's Theory of Cognitive Development

The ghost of Jean Piaget hangs over modern developmental psychology in a yet unresolved way. Piaget's theories provide a cogent overarching perspective on human cognitive development, coordinating broad theoretical ideas and diverse experimental results into a unified whole [Pia55]. Modern experimental work has shown Piaget's ideas to be often oversimplified and incorrect. However, what has replaced the Piagetan understanding is not an alternative unified and coherent theory, but a variety of microtheories addressing particular aspects of cognitive development. For this reason a number of contemporary theorists taking a computer science [Shu03] or dynamical systems [Wit07] approach to developmental psychology have chosen to adopt the Piagetan framework in spite of its demonstrated shortcomings, both because of its conceptual strengths and for lack of a coherent, more rigorously grounded alternative.

Our own position is that the Piagetan view of development has some fundamental truth to it, which is reflected via how nicely it fits with a broader view of development in complex systems. Indeed, Piaget viewed is developmental stages as as emerging from general "algebraic" principles rather than as being artifacts of the particulars of human psychology. But, Piaget's stages are probably best viewed as a general interpretive framework rather than a precise scientific theory. Our suspicion is that once the empirical science of developmental psychology has progressed further, it will become clearer how to fit the various data into a broad Piaget-like framework, perhaps differing in many details from what Piaget described in his works.

Piaget conceived of child development in four stages, each roughly identified with an age group, and corresponding closely to the system-theoretic stages mentioned above:

- **infantile**, corresponding to the automatic stage mentioned above
  - *Example:* Grasping blocks, piling blocks on top of each other, copying words that are heard
- **preoperational** and **concrete operational**, corresponding to the adaptive stage mentioned above
  - *Example:* Building complex blocks structures, from imagination and from imitating objects and pictures and based on verbal instructions; verbally describing what has been constructed
- **formal**, corresponding to the symbolic stage mentioned above
  - *Example:* Writing detailed instructions in words and diagrams, explaining how to construct particular structures out of blocks; figuring out general rules describing which sorts of blocks structures are likely to be most stable
- the reflexive stage mentioned above corresponds to what some post-Piagetan theorists have called the **post-formal** stage
  - *Example:* Using abstract lessons learned from building structures out of blocks to guide the construction of new ways to think and understand – "Zen and the art of blocks building" (by analogy to *Zen in the Art of Motorcycle Maintenance* [Pir84]).

More explicitly, Piaget defined his stages in psychological terms roughly as follows:

- **Infantile**: In this stage a mind develops basic world-exploration driven by instinctive actions. Reward-driven reinforcement of actions learned by imitation, simple associations between words and objects, actions and images, and the basic notions of time, space, and causality are developed. The most simple, practical ideas and strategies for action are learned.

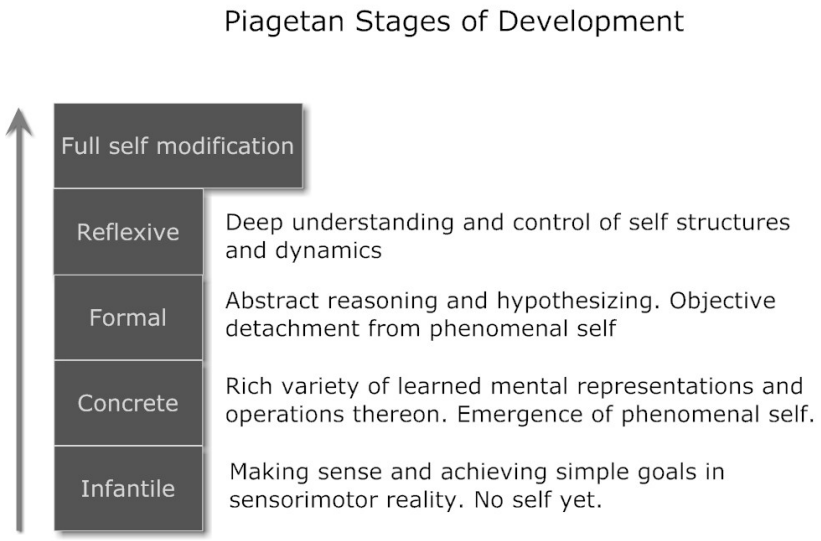


Fig. 11.1 Piagetan Stages of Cognitive Development

- **Preoperational**: At this stage we see the formation of mental representations, mostly poorly organized and un-abstracted, building mainly on intuitive rather than logical thinking. Word-object and image-object associations become systematic rather than occasional. Simple syntax is mastered, including an understanding of subject-argument relationships. One of the crucial learning achievements here is "object permanence"–infants learn that objects persist even when not observed. However, a number of cognitive failings persist with respect to reasoning about logical operations, and abstracting the effects of intuitive actions to an abstract theory of operations.
- **Concrete**: More abstract logical thought is applied to the physical world at this stage. Among the feats achieved here are: reversibility–the ability to undo steps already done; conservation–understanding that properties can persist in spite of appearances; theory of mind–an understanding of the distinction between what I know and what others know (If I cover my eyes, can you still see me?). Complex concrete operations, such as putting items in height order, are easily achievable. Classification becomes more sophisticated, yet the mind still cannot master purely logical operations based on abstract logical representations of the observational world.
- **Formal**: Abstract deductive reasoning, the process of forming, then testing hypotheses, and systematically reevaluating and refining solutions, develops at this stage, as does the ability to reason about purely abstract concepts without reference to concrete physical objects. This is adult human-level intelligence. Note that the capability for formal operations

is intrinsic in the PLN component of CogPrime , but in-principle capability is not the same as pragmatic, grounded, controllable capability.

Very early on, Vygotsky [Vyg86] disagreed with Piaget's explanation of his stages as inherent and developed by the child's own activities, and Piaget's prescription of good parenting as not interfering with a child's unfettered exploration of the world. Some modern theorists have critiqued Piaget's stages as being insufficiently socially grounded, and these criticisms trace back to Vygotsky's focus on the social foundations of intelligence, on the fact that children function in a world surrounded by adults who provide a cultural context, offering ongoing assistance, critique, and ultimately validation of the child's developmental activities.

Vygotsky also was an early critic of the idea that cognitive development is continuous, and continues beyond Piaget's formal stage. Gagne [RBW92] also believes in continuity, and that learning of prerequisite skills made the learning of subsequent skills easier and faster without regard to Piagetan stage formalisms. Subsequent researchers have argued that Piaget has merely constructed ad hoc descriptions of the sequential development of behaviour [Gib78, Bro84, CP05]. We agree that learning is a continuous process, and our notion of stages is more statistically constructed than rigidly quantized.

Critique of Piaget's notion of transitional "half stages" is also relevant to a more comprehensive hierarchical view of development. Some have proposed that Piaget's half stages are actually stages [Bro84]. As Commons and Pekker [CP05] point out: "the definition of a stage that was being used by Piaget was based on analyzing behaviors and attempting to impose different structures on them. There is no underlying logical or mathematical definition to help in this process . . . " Their Hierarchical Complexity development model uses task achievement rather than ad hoc stage definition as the basis for constructing relationships between phases of developmental ability–an approach which we find useful, though our approach is different in that we define stages in terms of specific underlying cognitive mechanisms.

Another critique of Piaget is that one individual's performance is often at different ability stages depending on the specific task (for example [GE86]). Piaget responded to early critiques along these lines by calling the phenomenon "horizontal décalage," but neither he nor his successors [Fis80, Cas85] have modified his theory to explain (rather than merely describe) it. Similarly to Thelen and Smith [TS94], we observe that the abilities encapsulated in the definition of a certain stage emerge gradually during the previous stage–so that the onset of a given stage represents the mastery of a cognitive skill that was previously present only in certain contexts.

Piaget also had difficulty accepting the idea of a preheuristic stage, early in the infantile period, in which simple trial-and-error learning occurs without significant heuristic guidance [Bic88], a stage which we suspect exists and allows formulation of heuristics by aggregation of learning from preheuristic pattern mining. Coupled with his belief that a mind's innate abilities at

birth are extremely limited, there is a troublingly unexplained transition from inability to ability in his model.

Finally, another limiting aspect of Piaget's model is that it did not recognize any stages beyond formal operations, and included no provisions for exploring this possibility. A number of researchers [Bic88, Arl75, CRK82, Rie73, Mar01] have described one or more postformal stages. Commons and colleagues have also proposed a task-based model which provides a framework for explaining stage discrepancies across tasks and for generating new stages based on classification of observed logical behaviors. [KK90] promotes a statistical conception of stage, which provides a good bridge between task-based and stage-based models of development, as statistical modeling allows for stages to be roughly defined and analyzed based on collections of task behaviors.

[CRK82] postulates the existence of a postformal stage by observing *elevated levels of abstraction* which, they argue, are not manifested in formal thought. [CTS+98] observes a postformal stage when subjects become capable of analyzing and coordinating complex logical systems with each other, creating metatheoretical supersystems. In our model, with the reflexive stage of development, we expand this definition of metasystemic thinking to include the ability to consciously refine one's own mental states and formalisms of thinking. Such self-reflexive refinement is necessary for learning which would allow a mind to analytically devise entirely new structures and methodologies for both formal and postformal thinking.

In spite of these various critiques and limitations, however, we have found Piaget's ideas very useful, and in Section 11.4 we will explore ways of defining them rigorously in the specific context of CogPrime 's declarative knowledge store and probabilistic logic engine.

11.3.1 Perry's Stages

Also relevant is William Perry's [Per70, Per81] theory of the stages ("positions" in his terminology) of intellectual and ethical development, which constitutes a model of iterative refinement of approach in the developmental process of coming to intellectual and ethical maturity. These stages, depicted in Table 11.2 form an analytical tool for discerning the modality of belief of an intelligence by describing common cognitive approaches to handling the complexities of real world ethical considerations.

Stage	Substages
Dualism / Received Knowledge [Infantile]	Basic duality ("All problems are solvable. I must learn the correct solutions.") Full dualism ("There are different, contradictory solutions to many problems. I must learn the correct solutions, and ignore the incorrect ones")
Multiplicity [Concrete]	Early multiplicity ("Some solutions are known, others aren't. I must learn how to find correct solutions.") Late Multiplicity: cognitive dissonance regarding truth. ("Some problems are unsolvable, some are a matter of personal taste, therefore I must declare my own intellectual path.")
Relativism / Procedural Knowledge [Formal]	Contextual Relativism ("I must learn to evaluate solutions within a context, and relative to supporting observation.") Pre-Commitment ("I must evaluate solutions, then commit to a choice of solution.")
Commitment / Constructed Knowledge [Formal / Reflexive]	Commitment ("I have chosen a solution.") Challenges to Commitment ("I have seen unexpected implications of my commitment, and the responsibility I must take.") Post-Commitment ("I must have an ongoing, nuanced relationship to the subject in which I evaluate each situation on a case-by-case basis with respects to its particulars rather than an ad-hoc application of unchallenged ideology.")

Table 11.2 Perry's Developmental Stages [with corresponding Piagetan Stages in brackets]

11.3.2 Keeping Continuity in Mind

Continuity of mental stages, and the fact that a mind may appear to be in multiple stages of development simultaneously (depending upon the tasks being tested), are crucial to our theoretical formulations and we will touch upon them again here. Piaget attempted to address continuity with the creation of transitional "half stages". We prefer to observe that each stage feeds into the other and the end of one stage and the beginning of the next blend together.

The distinction between formal and post-formal, for example, seems to "merely" be the application of formal thought to oneself. However, the distinction between concrete and formal is "merely" the buildup to higher levels of complexity of the classification, task decomposition, and abstraction capabilities of the concrete stage. The stages represent general trends in ability on a continuous curve of development, not discrete states of mind which are jumped-into quantum style after enough "knowledge energy" builds-up to cause the transition.

Observationally, this appears to be the case in humans. People learn things gradually, and show a continuous development in ability, not a quick jump from ignorance to mastery. We believe that this gradual development of ability is the signature of genuine learning, and that prescriptively an AGI system must be designed in order to have continuous and asymmetrical development across a variety of tasks in order to be considered a genuine learning system.

While quantum leaps in ability may be possible in an AGI system which can just "graft" new parts of brain onto itself (or an augmented human which may someday be able to do the same using implants), such acquisition of knowledge is not really learning. Grafting on knowledge does not build the cognitive pathways needed in order to actually learn. If this is the only mechanism available to an AGI system to acquire new knowledge, then it is not really a learning system.

11.4 Piaget's Stages in the Context of Uncertain Inference

Piaget's developmental stages are very general, referring to overall types of learning, not specific mechanisms or methods. This focus was natural since the context of his work was *human* developmental psychology, and neuroscience has not yet progressed to the point of understanding the neural mechanisms underlying any sort of inference (and certainly was nowhere near to doing so in Piaget's time!). But if one is studying developmental psychology in an AGI context where one knows something about the internal mechanisms of the AGI system under consideration, then one can work with a more specific model of learning. Our focus here is on AGI systems whose operations contain uncertain inference as a central component. Obviously the main focus is CogPrime but the essential ideas apply to any other uncertain inference centric AGI architecture as well.

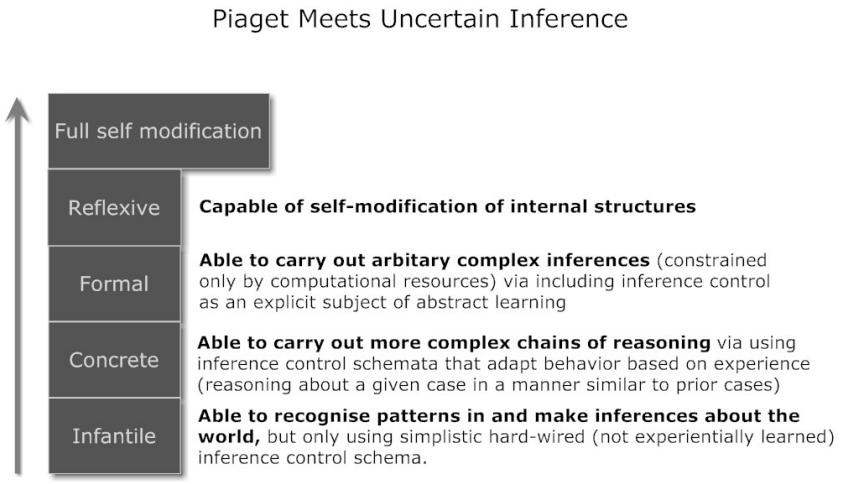


Fig. 11.2 Piagetan Stages of Development, as Manifested in the Context of Uncertain Inference

An uncertain inference system, as we consider it here, consists of four components, which work together in a feedback-control loop 11.3

1. a content representation scheme
2. an uncertainty representation scheme
3. a set of inference rules
4. a set of inference control schemata

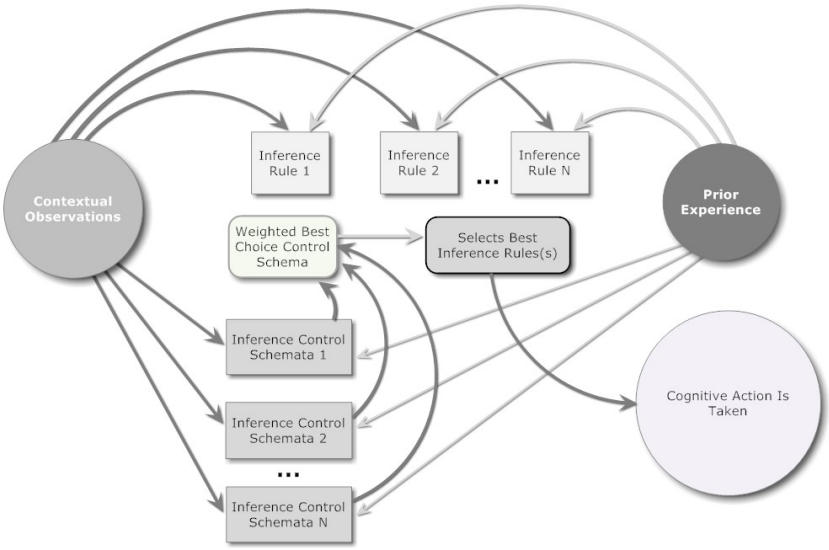


Fig. 11.3 A Simplified Look at Feedback-Control in Uncertain Inference

Broadly speaking, examples of content representation schemes are predicate logic and term logic [**?**]. Examples of uncertainty representation schemes are fuzzy logic [Zad78], imprecise probability theory [Goo86, FC86], Dempster-Shafer theory [Sha76, **?**], Bayesian probability theory [**?**], NARS [Wan95], and the Atom representation used in CogPrime , briefly alluded to in Chapter 6 above and described in depth in later chapters.

Many, but not all, approaches to uncertain inference involve only a limited, weak set of inference rules (e.g. not dealing with complex quantified expressions). CogPrime 's PLN inference framework, like NARS and some other uncertain inference frameworks, contains uncertain inference rules that apply to logical constructs of arbitrary complexity. Only a system capable of dealing with constructs of arbitrary (or at least very high) complexity will have any potential of leading to human-level, human-like intelligence.

The subtlest part of uncertain inference is inference control: the choice of which inferences to do, in what order. Inference control is the primary area in which human inference currently exceeds automated inference. Humans are

not very efficient or accurate at carrying out inference rules, with or without uncertainty, but we are very good at determining which inferences to do and in what order, in any given context. The lack of effective, context-sensitive inference control heuristics is why the general ability of current automated theorem provers is considerably weaker than that of a mediocre university mathematics major [Mac95].

We now review the Piagetan developmental stages from the perspective of AGI systems heavily based on uncertain inference.

11.4.1 The Infantile Stage

In this initial stage, the mind is able to recognize patterns in and conduct inferences about the world, but only using simplistic hard-wired (not experientially learned) inference control schema, along with pre-heuristic pattern mining of experiential data.

In the infantile stage an entity is able to recognize patterns in and conduct inferences about its sensory surround context (i.e., it's "world"), but only using simplistic, hard-wired (not experientially learned) inference control schemata. Preheuristic pattern-mining of experiential data is performed in order to build future heuristics about analysis of and interaction with the world.

Infantile stage tasks include:

1. Exploratory behavior in which useful and useless / dangerous behavior is differentiated by both trial and error observation, and by parental guidance.
2. Development of "habits" – i.e. Repeating tasks which were successful once to determine if they always / usually are so.
3. Simple goal-oriented behavior such as "find out what cat hair tastes like" in which one must plan and take several sequentially dependent steps in order to achieve the goal.

Inference control is very simple during the infantile stage (Figure 11.4), as it is the stage during which both the most basic knowledge of the world is acquired, and the most basic of cognition and inference control structures are developed as the building block upon which will be built the next stages of both knowledge and inference control.

Another example of a cognitive task at the borderline between infantile and concrete cognition is learning object permanence, a problem discussed in the context of CogPrime 's predecessor "Novamente Cognition Engine" system in [**?**]. Another example is the learning of word-object associations: e.g. learning that when the word "ball" is uttered in various contexts ("Get me the ball," "That's a nice ball," etc.) it generally refers to a certain type of object. The key point regarding these "infantile" inference problems, from the CogPrime

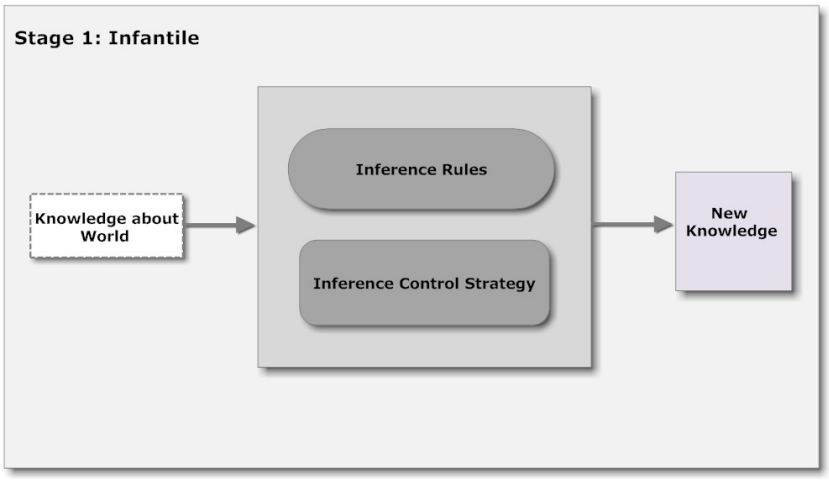


Fig. 11.4 Uncertain Inference in the Infantile Stage

perspective, is that assuming one provides the inference system with an appropriate set of perceptual and motor ConceptNodes and SchemaNodes, the chains of inference involved are short. They involve about a dozen inferences, and this means that the search tree of possible PLN inference rules walked by the PLN backward-chainer is relatively shallow. Sophisticated inference control is not required: standard AI heuristics are sufficient.

In short, textbook narrow-AI reasoning methods, utilized with appropriate uncertainty-savvy truth value formulas and coupled with appropriate representations of perceptual and motor inputs and outputs, correspond roughly to Piaget's infantile stage of cognition. The simplistic approach of these narrow-AI methods may be viewed as a method of creating building blocks for subsequent, more sophisticated heuristics.

In our theory Piaget's preoperational phase appears as transitional between the infantile and concrete operational phases.

11.4.2 The Concrete Stage

At this stage, the mind is able to carry out more complex chains of reasoning regarding the world, via using inference control schemata that adapt behavior based on experience (reasoning about a given case in a manner similar to prior cases).

In the concrete operational stage (Figure 11.5), an entity is able to carry out more complex chains of reasoning about the world. Inference control schemata which adapt behavior based on experience, using experientially

learned heuristics (including those learned in the prior stage), are applied to both analysis of and interaction with the sensory surround / world.

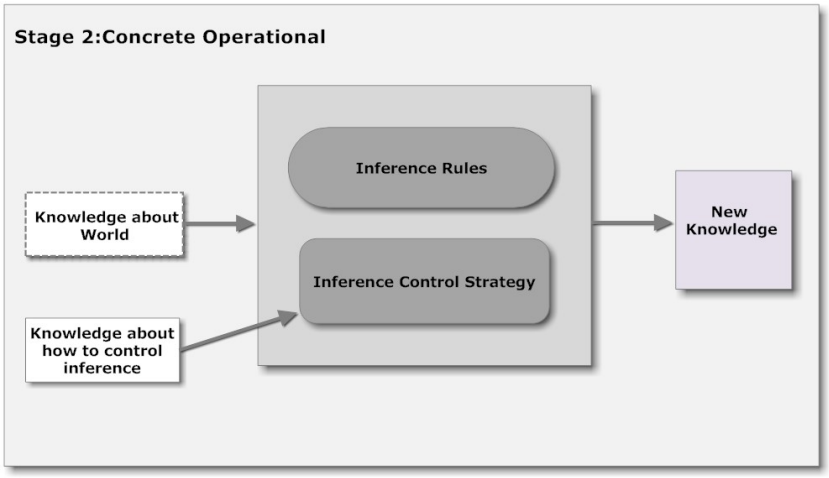


Fig. 11.5 Uncertain Inference in the Concrete Operational Stage

Concrete Operational stage tasks include:

1. Conservation tasks, such as conservation of number,
2. Decomposition of complex tasks into easier subtasks, allowing increasingly complex tasks to be approached by association with more easily understood (and previously experienced) smaller tasks,
3. Classification and Serialization tasks, in which the mind can cognitively distinguish various disambiguation criteria and group or order objects accordingly.

In terms of inference control this is the stage in which actual knowledge about how to control inference itself is first explored. This means an emerging understanding of inference itself as a cognitive task and methods for learning, which will be further developed in the following stages.

Also, in this stage a special cognitive task capability is gained: "Theory of Mind," which in cognitive science refers to the ability to understand the fact that not only oneself, but other sentient beings have memories, perceptions, and experiences. This is the ability to conceptually "put oneself in another's shoes" (even if you happen to assume incorrectly about them by doing so).

11.4.2.1 Conservation of Number

Conservation of number is an example of a learning problem classically categorized within Piaget's concrete-operational phase, a "conservation laws"

problem, discussed in [Shu03] in the context of software that solves the problem using (logic-based and neural net) narrow-AI techniques. Conservation laws are very important to cognitive development.

Conservation is the idea that a quantity remains the same despite changes in appearance. If you show a child some objects and then spread them out, an infantile mind will focus on the spread, and believe that there are now more objects than before, whereas a concrete-operational mind will understand that the quantity of objects has not changed.

Conservation of number seems very simple, but from a developmental perspective it is actually rather difficult. "Solutions" like those given in [Shu03] that use neural networks or customized logical rule-bases to find specialized solutions that solve only this problem fail to fully address the issue, because these solutions don't create knowledge adequate to aid with the solution of related sorts of problems.

We hypothesize that this problem is hard enough that for an inference-based AGI system to solve it in a developmentally useful way, its inferences must be guided by meta-inferential lessons learned from prior similar problems. When approaching a number conservation problem, for example, a reasoning system might draw upon past experience with set-size problems (which may be trial-and-error experience). This is not a simple "machine learning" approach whose scope is restricted to the current problem, but rather a heuristically guided approach which (a) aggregates information from prior experience to guide solution formulation for the problem at hand, and (b) adds the present experience to the set of relevant information about quantification problems for future refinement of thinking.

Fig. 11.6 Conservation of Number

For instance, a very simple context-specific heuristic that a system might learn would be: "When evaluating the truth value of a statement related to the number of objects in a set, it is generally not that useful to explore branches of the backwards-chaining search tree that contain relationships regarding the sizes, masses, or other physical properties of the objects in the set." This heuristic itself may go a long way toward guiding an inference process toward a correct solution to the problem–but it is not something that a mind needs to know "a priori." A concrete-operational stage mind may learn this by data-mining prior instances of inferences involving sizes of sets. Without such experience-based heuristics, the search tree for such a problem will likely be unacceptably large. Even if it is "solvable" without such heuristics, the solutions found may be overly fit to the particular problem and not usefully generalizable.

11.4.2.2 Theory of Mind

Consider this experiment: a preoperational child is shown her favorite "Dora the Explorer" DVD box. Asked what show she's about to see, she'll answer "Dora." However, when her parent plays the disc, it's "Spongebob Squarepants." If you then ask her what show her friend will expect when given the "Dora" DVD box, she will respond "Spongebob" although she just answered "Dora" for herself. A child lacking a theory of mind can not reason through what someone else would think given knowledge other than her own current knowledge. Knowledge of self is intrinsically related to the ability to differentiate oneself from others, and this ability may not be fully developed at birth.

Several theorists [**?**, Fod94], based in part on experimental work with autistic children, perceive theory of mind as embodied in an innate module of the mind activated at a certain developmental stage (or not, if damaged). While we consider this possible, we caution against adopting a simplistic view of the "innate vs. acquired" dichotomy: if there is innateness it may take the form of an innate predisposition to certain sorts of learning [EBJ+97].

Davidson [Dav84], Dennett [Den87] and others support the common belief that theory of mind is dependent upon linguistic ability. A major challenge to this prevailing philosophical stance came from Premack and Woodruff [PW78] who postulated that prelinguistic primates do indeed exhibit "theory of mind" behavior. While Premack and Woodruff's experiment itself has been challenged, their general result has been bolstered by follow-up work showing similar results such as [TC97]. It seems to us that while theory of mind depends on many of the same inferential capabilities as language learning, it is not intrinsically dependent on the latter.

There is a school of thought often called the *Theory Theory* [BW88, Car85, Wel90] holding that a child's understanding of mind is best understood in terms of the process of iteratively formulating and refuting a series of naive theories about others. Alternately, Gordon [Gor86] postulates that theory of mind is related to the ability to run cognitive simulations of others' minds using one's own mind as a model. We suggest that these two approaches are actually quite harmonious with one another. In an uncertain AGI context, both theories and simulations are grounded in collections of uncertain implications, which may be assembled in context-appropriate ways to form theoretical conclusions or to drive simulations. Even if there is a special "mind-simulator" dynamic in the human brain that carries out simulations of other minds in a manner fundamentally different from explicit inferential theorizing, the inputs to and the behavior of this simulator may take inferential form, so that the simulator is in essence a way of efficiently and implicitly producing uncertain inferential conclusions from uncertain premises.

We have thought through the details by CogPrime system should be able to develop theory of mind via embodied experience, though at time of writing practical learning experiments in this direction have not yet been done. We

have not yet explored in detail the possibility of giving CogPrime a special, elaborately engineered "mind-simulator" component, though this would be possible; instead we have initially been pursuing a more purely inferential approach.

First, it is very simple for a CogPrime system to learn patterns such as "If I rotated by pi radians, I would see the yellow block." And it's not a big leap for PLN to go from this to the recognition that "You look like me, and you're rotated by pi radians relative to my orientation, therefore you probably see the yellow block." The only nontrivial aspect here is the "you look like me" premise.

Recognizing "embodied agent" as a category, however, is a problem fairly similar to recognizing "block" or "insect" or "daisy" as a category. Since the CogPrime agent can perceive most parts of its own "robot" body–its arms, its legs, etc.–it should be easy for the agent to figure out that physical objects like these look different depending upon its distance from them and its angle of observation. From this it should not be that difficult for the agent to understand that it is naturally grouped together with other embodied agents (like its teacher), not with blocks or bugs.

The only other major ingredient needed to enable theory of mind is "reflection"– the ability of the system to explicitly recognize the existence of knowledge in its own mind (note that this term "reflection" is not the same as our proposed "reflexive" stage of cognitive development). This exists automatically in CogPrime , via the built-in vocabulary of elementary procedures supplied for use within SchemaNodes (specifically, the atTime and TruthValue operators). Observing that "at time T, the weight of evidence of the link L increased from zero" is basically equivalent to observing that the link L was created at time T.

Then, the system may reason, for example, as follows (using a combination of several PLN rules including the above-given deduction rule):

Implication
    My eye is facing a block and it is not dark
    A relationship is created describing the block's color
Similarity
    My body
    My teacher's body
|-
Implication
    My teacher's eye is facing a block and it is not dark
    A relationship is created describing the block's color

This sort of inference is the essence of Piagetan "theory of mind." Note that in both of these implications the created relationship is represented as a variable rather than a specific relationship. The cognitive leap is that in the latter case the relationship actually exists in the teacher's implicitly hypothe-

sized mind, rather than in CogPrime 's mind. No explicit hypothesis or model of the teacher's mind need be created in order to form this implication–the hypothesis is created implicitly via inferential abstraction. Yet, a collection of implications of this nature may be used via an uncertain reasoning system like PLN to create theories and simulations suitable to guide complex inferences about other minds.

From the perspective of developmental stages, the key point here is that in a CogPrime context this sort of inference is too complex to be viably carried out via simple inference heuristics. This particular example must be done via forward chaining, since the big leap is to actually think of forming the implication that concludes inference. But there are simply too many combinations of relationships involving CogPrime 's eye, body, and so forth for the PLN component to viably explore all of them via standard forward-chaining heuristics. Experience-guided heuristics are needed, such as the heuristic that if physical objects A and B are generally physically and functionally similar, and there is a relationship involving some part of A and some physical object R, it may be useful to look for similar relationships involving an analogous part of B and objects similar to R. This kind of heuristic may be learned by experience–and the masterful deployment of such heuristics to guide inference is what we hypothesize to characterize the concrete stage of development. The "concreteness" comes from the fact that inference control is guided by analogies to prior similar situations.

11.4.3 The Formal Stage

In the formal stage, as shown in Figure 11.7, an agent should be able to carry out arbitrarily complex inferences (constrained only by computational resources, rather than by fundamental restrictions on logical language or form) via including inference control as an explicit subject of abstract learning.

In the formal stage, an entity is able to carry out arbitrarily complex inferences (constrained only by computational resources). Abstraction and inference about both the sensorimotor surround (world) and about abstract ideals themselves (including the final stages of indirect learning about inference itself) are fully developed.

Formal stage evaluation tasks are centered entirely around abstraction and higher-order inference tasks such as:

1. Mathematics and other formalizations.
2. Scientific experimentation and other rigorous observational testing of abstract formalizations.
3. Social and philosophical modeling, and other advanced applications of empathy and the Theory of Mind.

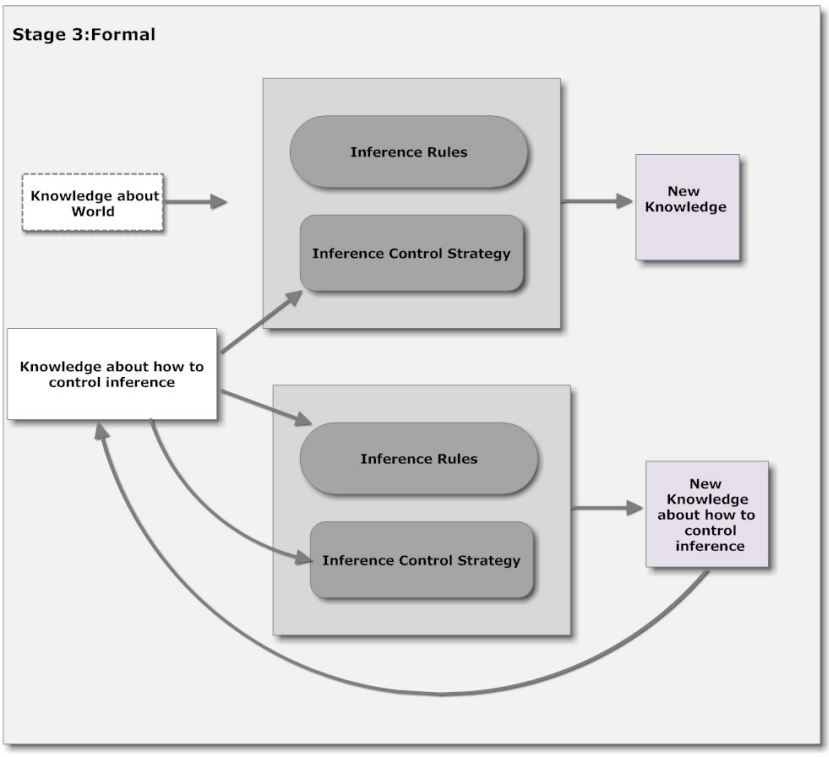


Fig. 11.7 Uncertain Inference in the Formal Stage

In terms of inference control this stage sees not just perception of new knowledge about inference control itself, but inference controlled reasoning about that knowledge and the creation of abstract formalizations about inference control which are reasoned-upon, tested, and verified or debunked.

11.4.3.1 Systematic Experimentation

The Piagetan formal phase is a particularly subtle one from the perspective of uncertain inference. In a sense, AGI inference engines already have strong capability for formal reasoning built in. Ironically, however, no existing inference engine is capable of deploying its reasoning rules in a powerfully effective way, and this is because of the lack of inference control heuristics adequate for controlling abstract formal reasoning. These heuristics are what arise during Piaget's formal stage, and we propose that in the content of uncertain inference systems, they involve the application of inference itself to the problem of refining inference control.

A problem commonly used to illustrate the difference between the Piagetan concrete operational and formal stages is that of figuring out the rules for making pendulums swing quickly versus slowly [IP58]. If you ask a child in the formal stage to solve this problem, she may proceed to do a number of experiments, e.g. build a long string with a light weight, a long string with a heavy weight, a short string with a light weight and a short string with a heavy weight. Through these experiments she may determine that a short string leads to a fast swing, a long string leads to a slow swing, and the weight doesn't matter at all.

The role of experiments like this, which test "extreme cases," is to make cognition easier. The formal-stage mind tries to map a concrete situation onto a maximally simple and manipulable set of abstract propositions, and then reason based on these. Doing this, however, requires an automated and instinctive understanding of the reasoning process itself. The above-described experiments are good ones for solving the pendulum problem because they provide data that is very easy to reason about. From the perspective of uncertain inference systems, this is the key characteristic of the formal stage: formal cognition approaches problems in a way explicitly calculated to yield tractable inferences.

Note that this is quite different from saying that formal cognition involves abstractions and advanced logic. In an uncertain logic-based AGI system, even infantile cognition may involve these–the difference lies in the level of inference control, which in the infantile stage is simplistic and hard-wired, but in the formal stage is based on an understanding of what sorts of inputs lead to tractable inference in a given context.

11.4.4 The Reflexive Stage

In the reflexive stage (Figure 11.8), an intelligent agent is broadly capable of self-modifying its internal structures and dynamics.

As an example in the human domain: highly intelligent and self-aware adult humans may carry out reflexive cognition by explicitly reflecting upon their own inference processes and trying to improve them. An example is the intelligent improvement of uncertain-truth-value-manipulation formulas. It is well demonstrated that even educated humans typically make numerous errors in probabilistic reasoning [GGK]. Most people don't realize it and continue to systematically make these errors throughout their lives. However, a small percentage of individuals make an explicit effort to increase their accuracy in making probabilistic judgments by consciously endeavoring to internalize the rules of probabilistic inference into their automated cognition processes.

In the uncertain inference based AGI context, what this means is: In the reflexive stage an entity is able to include inference control itself as an explicit

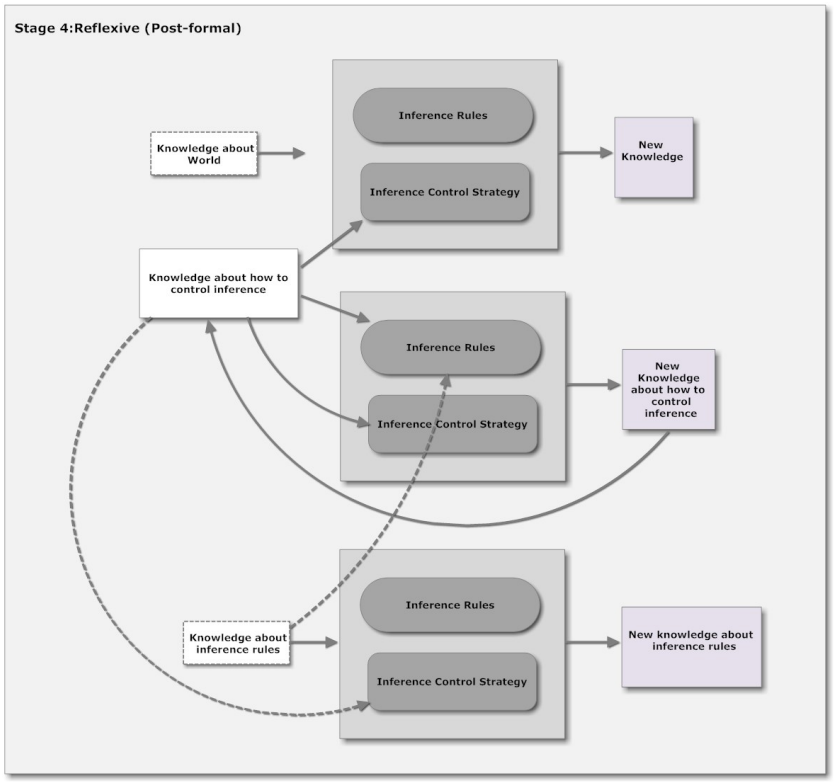


Fig. 11.8 The Reflexive Stage

subject of abstract learning (i.e. the ability to reason about one's own tactical and strategic approach to modifying one's own learning and thinking, and modify these inference control strategies based on analysis of experience with various cognitive approaches.

Ultimately, the entity can self-modify its internal cognitive structures. Any knowledge or heuristics can be revised, including metatheoretical and metasystemic thought itself. Initially this is done indirectly, but at least in the case of AGI systems it is theoretically possible to also do so directly. This might be considered as a separate stage of Full Self Modification, or else as the end phase of the reflexive stage. In the context of logical reasoning, self modification of inference control itself is the primary task in this stage. In terms of inference control this stage adds an entire new feedback loop for reasoning about inference control itself, as shown in Figure 11.8.

As a very concrete example, in later chapters we will see that, while PLN is founded on probability theory, it also contains a variety of heuristic assumptions that inevitably introduce a certain amount of error into its inferences. For example, PLN's probabilistic deduction embodies a heuristic in-

dependence assumption. Thus PLN contains an alternate deduction formula called the "concept geometry formula" that is better in some contexts, based on the assumption that ConceptNodes embody concepts that are roughly spherically-shaped in attribute space. A highly advanced CogPrime system could potentially augment the independence-based and concept-geometry-based deduction formulas with additional formulas of its own derivation, optimized to minimize error in various contexts. This is a simple and straightforward example of reflexive cognition–it illustrates the power accessible to a cognitive system that has formalized and reflected upon its own inference processes, and that possesses at least some capability to modify these.

In general, AGI systems can be expected to have much broader and deeper capabilities for self-modification than human beings. Ultimately it may make sense to view the AGI systems we implement as merely "initial conditions" for ongoing self-modification and self-organization. Chapter **??** discusses some of the potential technical details underlying this sort of thoroughgoing AGI self-modification.

Chapter 12
The Engineering and Development of Ethics

Co-authored with Stephan Vladimir Bugaj and Joel Pitt

12.1 Introduction

Most commonly, if a work on advanced AI mentions ethics at all, it occurs in a final summary chapter, discussing in broad terms some of the possible implications of the technical ideas presented beforehand. It's no coincidence that the order is reversed here: in the case of CogPrime , AGI-ethics considerations played a major role in the design process ... and thus the chapter on ethics occurs near the beginning rather than the end. In the CogPrime approach, ethics is not a particularly distinct topic, being richly interwoven with cognition and education and other aspects of the AGI project.

The ethics of advanced AGI is a complex issue with multiple aspects. Among the many issues there are:

1. Risks posed by the possibility of human beings using AGI systems for evil ends
2. Risks posed by AGI systems created without well-defined ethical systems
3. Risks posed by AGI systems with initially well-defined and sensible ethical systems eventually going rogue – an especially big risk if these systems are more generally intelligent than humans, and possess the capability to modify their own source code
4. the ethics of experimenting on AGI systems when one doesn't understand the nature of their experience
5. AGI rights: in what circumstances does using an AGI as a tool or servant constitute "slavery"

In this chapter we will focus mainly (though not exclusively) on the question of *how to create an AGI with a rational and beneficial ethical system.* After a somewhat wide-ranging discussion, we will conclude with eight general points that we believe should be followed in working toward "Friendly AGI" – most of which have to do, not with the internal design of the AGI, but with the way the AGI is taught and interfaced with the real world.

While most of the particulars discussed in this book have nothing to do with ethics, it's important for the reader to understand that AGI-ethics considerations have played a major role in many of our design decisions, underlying much of the technical contents of the book. As the materials in this chapter should make clear, ethicalness is probably not something that one can meaningfully tack onto an AGI system at the end, after developing the rest – it is likely infeasible to architect an intelligent agent and then add on an "ethics module." Rather, ethics is something that has to do with all the different memory systems and cognitive processes that constitute an intelligent system – and it's something that involves both cognitive architecture *and* the exploration a system does and the instruction it receives. It's a very complex matter that is richly intermixed with all the other aspects of intelligence, and here we will treat it as such.

12.2 Review of Current Thinking on the Risks of AGI

Before proceeding to outline our own perspective on AGI ethics in the context of CogPrime , we will review the main existing strains of thought on the potential ethical dangers associated with AGI. One SF film after another has highlighted these dangers, lodging the issue deep in our cultural awareness; unsurprisingly, much less attention has been paid to serious analysis of the risks in their various dimensions, but there is still a non-trivial literature worth paying attention to.

Hypothetically, an AGI with superhuman intelligence and capability could dispense with humanity altogether – i.e. posing an "existential risk" [Bos02]. In the worst case, an evil but brilliant AGI, perhaps programmed by a human sadist, could consign humanity to unimaginable tortures (i.e. realizing a modern version of the medieval Christian fantasies of hell). On the other hand, the potential benefits of powerful AGI also go literally beyond human imagination. It seems quite plausible that an AGI with massively superhuman intelligence and positive disposition toward humanity could provide us with truly dramatic benefits, such as a virtual end to material scarcity, disease and aging. Advanced AGI could also help individual humans grow in a variety of directions, including directions leading beyond "legacy humanity," according to their own taste and choice.

Eliezer Yudkowsky has introduced the term "Friendly AI", to refer to advanced AGI systems that act with human benefit in mind [Yud06]. Exactly what this means has not been specified precisely, though informal interpretations abound. Goertzel [Goe06b] has sought to clarify the notion in terms of three core values of Joy, Growth and Freedom. In this view, a Friendly AI would be one that advocates individual and collective human joy and growth, while respecting the autonomy of human choices.

Some (for example, Hugo de Garis, [DG05]), have argued that Friendly AI is essentially an impossibility, in the sense that the odds of a dramatically superhumanly intelligent mind worrying about human benefit are vanishingly small. If this is the case, then the best options for the human race would presumably be to either avoid advanced AGI development altogether, or to else fuse with AGI before it gets too strongly superhuman, so that beings-originated-as-humans can enjoy the benefits of greater intelligence and capability (albeit at cost of sacrificing their humanity).

Others (e.g. Mark Waser [Was09]) have argued that Friendly AI is essentially inevitable, because greater intelligence correlates with greater morality. Evidence from evolutionary and human history is adduced in favor of this point, along with more abstract arguments.

Yudkowsky [Yud06] has discussed the possibility of creating AGI architectures that are in some sense "provably Friendly" – either mathematically, or else at least via very tight lines of rational verbal argumentation. However, several issues have been raised with this approach. First, it seems likely that proving mathematical results of this nature would first require dramatic advances in multiple branches of mathematics. Second, such a proof would require a formalization of the goal of "Friendliness," which is a subtler matter than it might seem [Leg06b, Leg06a]. Formalization of human morality has vexed moral philosophers for quite some time. Finally, it is unclear the extent to which such a proof could be created in a generic, environment-independent way – but if the proof depends on properties of the physical environment, then it would require a formalization of the environment itself, which runs up against various problems such as the complexity of the physical world and also the fact that we currently have no complete, consistent theory of physics. Kaj Sotala has provided a list of 14 objections to the Friendly AI concept, and suggested to answers to each of them [Sot11]. Stephen Omohundro [Omo08] has argued that any advanced AI system will very likely demonstrate certain "basic AI drives", such as desiring to be rational, to self-protect, to acquire resources, and to preserve and protect its utility function and avoid counterfeit utility; these drives, he suggests, must be taken carefully into account in formulating approaches to Friendly AI.

The problem of formally or at least very carefully defining the goal of Friendliness has been considered from a variety of perspectives, none showing dramatic success. Yudkowsky [Yud04] has suggested the concept of "Coherent Extrapolated Volition", which roughly refers to the extrapolation of the common values of the human race. Many subtleties arise in specifying this concept – e.g. if Bob Jones is often possessed by a strong desire to kill all Martians, but he deeply aspires to be a nonviolent person, then the CEV approach would not rate "killing Martians" as part of Bob's contribution to the CEV of humanity.

Goertzel [Goe10a] has proposed a related notion of Coherent Aggregated Volition (CAV), which eschews the subtleties of extrapolation, and simply seeks a reasonably *compact, coherent, consistent* set of values that is fairly

close to the collective value-set of humanity. In the CAV approach, "killing Martians" would be removed from humanity's collective value-set because it's uncommon and not part of the most compact/coherent/consistent overall model of human values, rather than because of Bob Jones's aspiration to nonviolence.

One thought we have recently entertained is that the core concept underlying CAV might be better thought of as CBV or "Coherent Blended Volition." CAV seems to be easily misinterpreted as meaning the average of different views, which was not the original intention. The CBV terminology clarifies that the CBV of a diverse group of people should not be thought of as an average of their perspectives, but as something more analogous to a "conceptual blend" [FT02] – incorporating the most essential elements of their divergent views into a whole that is overall compact, elegant and harmonious. The subtlety here (to which we shall return below) is that for a CBV blend to be broadly acceptable, the different parties whose views are being blended must agree to some extent that enough of the essential elements of their own views have been included. The process of arriving at this sort of consensus may involve extrapolation of a roughly similar sort to that considered in CEV.

Multiple attempts at axiomatization of human values have also been attempted, e.g. with a view toward providing near-term guidance to military robots (see e.g. Arkin's excellent though chillingly-titled book *Governing Lethal Behavior in Autonomous Robots* [Ark09b], the result of US military funded research). However, there are reasonably strong arguments that human values (similarly to e.g. human language or human perceptual classification rules) are too complex and multifaceted to be captured in any compact set of formal logic rules. Wallach [WA10] has made this point eloquently, and argued the necessity of fusing top-down (e.g. formal logic based) and bottom-up (e.g. self-organizing learning based) approaches to machine ethics.

A number of more sociological considerations also arise. It is sometimes argued that the risk from highly-advanced AGI going morally awry on its own may be less than that of moderately-advanced AGI being used by human beings to advocate immoral ends. This possibility gives rise to questions about the ethical value of various practical modalities of AGI development, for instance:

- Should AGI be developed in a top-secret installation by a select group of individuals selected for a combination of technical and scientific brilliance and moral uprightness, or other qualities deemed relevant (a "closed approach")? Or should it be developed out in the open, in the manner of open-source software projects like Linux? (an "open approach"). The open approach allows the collective intelligence of the world to more fully participate – but also potentially allows the more unsavory elements of the human race to take some of the publicly-developed AGI concepts and tools private, and develop them into AGIs with selfish or evil purposes in mind. Is there some meaningful intermediary between these extremes?

- Should governments regulate AGI, with Friendliness in mind (as advocated carefully by e.g Bill Hibbard [Hib02])? Or will this just cause AGI development to move to the handful of countries with more liberal policies? ... or cause it to move underground, where nobody can see the dangers developing? As a rough analogue, it's worth noting that the US government's imposition of restrictions on stem cell research, under President George W. Bush, appears to have directly stimulated the provision of additional funding for stem cell research in other nations like Korea, Singapore and China.

The former issue is, obviously, highly relevant to CogPrime (which is currently being developed via the open source CogPrime project); and so the various dimensions of this issues are worth briefly sketching here.

We have a strong skepticism of self-appointed elite groups that claim (even if they genuinely believe) that they know what's best for everyone, and a healthy respect for the power of collective intelligence and the Global Brain, which the open approach is ideal for tapping. On the other hand, we also understand the risk of terrorist groups or other malevolent agents forking an open source AGI project and creating something terribly dangerous and destructive. Balancing these factors against each other rigorously, seems beyond the scope of current human science.

Nobody really understands the social dynamics by which open technological knowledge plays out in our *current* world, let alone hypothetical future scenarios. Right now there exists open knowledge about many very dangerous technologies, and there exist many terrorist groups, yet these groups fortunately make scant use of these technologies. The reasons why appear to be essentially sociological – the people involved in these terrorist groups tend not to be the ones who have mastered the skills of turning public knowledge on cutting-edge technologies into real engineered systems. But while it's easy to observe this sociological phenomenon, we certainly have no way to estimate its *quantitative extent* from first principles. We don't really have a strong understanding of how safe we are right now, given the technology knowledge available right now via the Internet, textbooks, and so forth. Even relatively straightforward issues such as nuclear proliferation remain confusing, even to the experts.

It's also quite clear that keeping powerful AGI locked up by an elite group doesn't really provide reliable protection against malevolent human agents. History is rife with such situations going awry, e.g. by the leadership of the group being subverted, or via brute force inflicted by some outside party, or via a member of the elite group defecting to some outside group in the interest of personal power or reward or due to group-internal disagreements, etc. There are many things that can go wrong in such situations, and the confidence of any particular group that they are immune to such issues, cannot be taken very seriously. Clearly, neither the open nor closed approach qualifies as a panacea.

12.3 The Value of an Explicit Goal System

One of the subtle issues confronted in the quest to design ethical AGIs is how closely one wants to emulate human ethical judgment and behavior. Here one confronts the brute fact that, even according to their own deeply-held standards, humans are not all that ethical. One high-level conclusion we came to very early in the process of designing CogPrime is that, just as humans are not the most intelligent minds achievable, they are also not the most ethical minds achievable. Even if one takes human ethics, broadly conceived, as the standard – there are almost surely possible AGI systems that are much more ethical *according to human standards* than nearly all human beings. This is not mainly because of ethics-specific features of the human mind, but rather because of the nature of the human motivational system, which leads to many complexities that drive humans to behaviors that are unethical according to their own standards. So, one of the design decisions we made for CogPrime – with ethics as well as other reasons in mind – was *not* to closely imitate the human motivational system, but rather to craft a novel motivational system combining certain aspects of the human motivational system with other profoundly non-human aspects.

On the other hand, the design of ethical AGI systems still has a lot to gain from the study of human ethical cognition and behavior. Human ethics has many aspects, which we associate here with the different types of memory, and it's important that AGI systems can encompass all of them. Also, as we will note below, human ethics develops in childhood through a series of natural stages, parallel to and entwined with the cognitive developmental stages reviewed in Chapter 11 above. We will argue that for an AGI with a virtual or robotic body, it makes sense to think of ethical development as proceeding through similar stages. In a CogPrime context, the particulars of these stages can then be understood in terms of the particulars of CogPrime 's cognitive processes – which brings AGI ethics from the domain of theoretical abstraction into the realm of practical algorithm design and education.

But even if the human stages of ethical development make sense for non-human AGIs, this doesn't mean the particulars of the human motivational system need to be replicated in these AGIs, regarding ethics or other matters. A key point here is that, in the context of human intelligence, the concept of a "goal" is a descriptive abstraction. But in the AGI context, it seems quite valuable to introduce goals as explicit design elements (which is what is done in CogPrime) – both for ethical reasons and for broader AGI design reasons.

Humans may adopt goals for a time and then drop them, may pursue multiple conflicting goals simultaneously, and may often proceed in an apparently goal-less manner. Sometimes the goal that a person appears to be pursuing, may be very different than the one they think they're pursuing. Evolutionary psychology [BDL93] argues that, directly or indirectly, all humans are ultimately pursuing the goal of maximizing the inclusive fitness of their genes – but given the complex mix of evolution and self-organization

in natural history [Sal93], this is hardly a general explanation for human behavior. Ultimately, in the human context, "goal" is best thought of as a frequently useful heuristic concept.

AGI systems, however, need not emulate human cognition in every aspect, and may be architected with explicit "goal systems." This provides no guarantee that said AGI systems will actually pursue the goals that their goal systems specify – depending on the role that the goal system plays in the overall system dynamics, sometimes other dynamical phenomena might intervene and cause the system to behave in ways opposed to its explicit goals. However, we submit that this design sketch provides a better framework than would exist in an AGI system closely emulating the human brain.

We realize this point may be somewhat contentious – a counter-argument would be that the human brain is known to support at least *moderately* ethical behavior, according to human ethical standards, whereas less brain-like AGI systems are much less well understood. However, the obvious counter-counterpoints are that:

- Humans are not all that consistently ethical, so that creating AGI systems potentially much more practically powerful than humans, but with closely humanlike ethical, motivational and goal systems, could in fact be quite dangerous
- The effect on a human-like ethical/motivational/goal system of increasing the intelligence, or changing the physical embodiment or cognitive capabilities, of the agent containing the system, is unknown and difficult to predict given all the complexities involved

The course we tentatively recommend, and are following in our own work, is to develop AGI systems with explicit, hierarchically-dominated goal systems. That is:

- create one or more "top goals" (we call them Ubergoals in CogPrime)
- have the system derive subgoals from these, using its own intelligence, potentially guided by educational interaction or explicit programming
- have a significant percentage of the system's activity governed by the explicit pursuit of these goals

Note that the "significant percentage" need not be 100%; CogPrime , for example, combines explicitly goal-directed activity with other "spontaneous" activity. Requiring that all activity be explicitly goal-directed may be too strict a requirement to place on AGI architectures.

The next step, of course, is for the top-level goals to be chosen in accordance with the principle of human-Friendliness. The next one of our eight points, about the Global Brain, addresses one way of doing this. In our near-term work with CogPrime , we are using simplistic approaches, with a view toward early-stage system testing.

12.4 Ethical Synergy

An explicit goal system provides an explicit way to ensure that ethical principles (as represented in system goals) play a significant role in guiding an AGI system's behavior. However, in an integrative design like CogPrime the goal system is only a small part of the overall story, and it's important to also understand how ethics relates to the other aspects of the cognitive architecture.

One of the more novel ideas presented in this chapter is that different types of ethical intuition may be associated with different types of memory – and to possess mature ethics, a mind must display *ethical synergy* between the ethical processes associated with its memory types. Specifically, we suggest that:

- **Episodic memory** corresponds to the process of ethically assessing a situation based on similar prior situations
- **Sensorimotor memory** corresponds to "mirror neuron" type ethics, where you feel another person's feelings via mirroring their physiological emotional responses and actions
- **Declarative memory** corresponds to rational ethical judgment
- **Procedural memory** corresponds to "ethical habit" ... learning by imitation and reinforcement to do what is right, even when the reasons aren't well articulated or understood
- **Attentional memory** corresponds to the existence of appropriate patterns guiding one to pay adequate attention to ethical considerations at appropriate times
- **Intentional memory** corresponds to the pervasion of ethics through one's choices about subgoaling (which leads into "when do the ends justify the means" ethical-balance questions)

One of our suggestions regarding AGI ethics is that an ethically mature person or AGI must both master and balance all these kinds of ethics. We will focus especially here on declarative ethics, which corresponds to Kohlberg's theory of logical ethical judgment; and episodic ethics, which corresponds to Gilligan's theory of empathic ethical judgment. Ultimately though, all five aspects are critically important; and a CogPrime system if appropriately situated and education should be able to master and integrate all of them.

12.4.1 Stages of Development of Declarative Ethics

Complementing generic theories of cognitive development such as Piaget's and Perry's, theorists have also proposed specific stages of moral and ethical development. The two most relevant theories in this domain are those of

Kohlberg and Gilligan, which we will review here, both individually and in terms of their integration and application in the AGI context.

Lawrence Kohlberg's [**?**, **?**] moral development model, called the "ethics of justice" by Gilligan, is based on a rational modality as the central vehicle for moral development. In our perspective this is a firmly *declarative* form of ethics, based on explicit analysis and reasoning. It is based on an impartial regard for persons, proposing that ethical consideration must be given to all individual intelligences without a priori judgment (prejudice). Consideration is given for individual merit and preferences, and the goals of an ethical decision are equal treatment (in the general, not necessarily the particular) and reciprocity. Echoing Kant's [**?**] categorical imperative, the decisions considered most successful in this model are those which exhibit "reversibility", where a moral act within a particular situation is evaluated in terms of whether or not the act would be satisfactory even if particular persons were to switch roles within the situation. In other words, a situational, contextualized "do unto others as you would have them do unto you" criterion. The ethics of justice can be viewed as three stages (each of which has six substages, on which we will not elaborate here), depicted in Table 12.1.

Stage	**Substages**
Pre-Conventional	• Obedience and Punishment Orientation • Self-interest orientation
Conventional	• Interpersonal accord (conformity) orientation • Authority and social-order maintaining (law and order) orientation
Post-Conventional	• Social contract (human rights) orientation • Universal ethical principles (universal human rights) orientation

Table 12.1 Kohlberg's Stages of Development of the Ethics of Justice

In Kohlberg's perspective, cognitive development level contributes to moral development, as moral understanding emerges from increased cognitive capability in the area of ethical decision making in a social context. Relatedly, Kohlberg also looks at stages of social perspective and their consequent interpersonal outlook. As shown in Table 12.1, these are correlated to the stages of moral development, but also map onto Piagetian models of cognitive development (as pointed out e.g. by Gibbs [Gib78], who presents a modification/interpretation of Kohlberg's ideas intended to align them more

closely with Piaget's). Interpersonal outlook can be understood as rational understanding of the psychology of other persons (a theory of mind, with or without empathy). Stage One, emergent from the infantile congitive stage, is entirely selfish as only self awareness has developed. As cognitive sophistication about ethical considerations increases, so do the moral and social perspective stages. Concrete and formal cognition bring about the first instrumental egoism, and then social relations and systems perspectives, and from formal and then reflexive thinking about ethics comes the post-conventional modalities of contractualism and universal mutual respect.

Stage of Social Perspective	**Interpersonal Outlook**
Blind egoism	No interpersonal perspective. Only self is considered.
Instrumental egoism	See that others have goals and perspectives, and either conform to or rebel against norms.
Social Relationships perspective	Able to see abstract normative systems
Social Systems perspective	Recognize positive and negative intentions
Contractual perspective	Recognize that contracts (mutually beneficial agreements of any kind) will allow intelligences to increase the welfare of both.
Universal principle of mutual respect	See how human fallibility and frailty are impacted by communication.

Table 12.2 Kohlberg's Stages of Development of Social Perspective and Interpersonal Morals

12.4.1.1 Uncertain Inference and the Ethics of Justice

Taking our cue from the analysis given in Chapter 11 of Piagetan stages in uncertain inference based AGI systems (such as CogPrime), we may explore the manifestation of Kohlberg's stages in AGI systems of this nature. Uncertain inference seems generally well-suited as a declarative-ethics learning system, due to the nuanced ethical environment of real world situations. Probabilistic knowledge networks can model belief networks, imitative reinforcement learning based ethical pedagogy, and even simplistic moral maxims. In principle, they have the flexibility to deal with complex ethical decisions, including not only weighted "for the greater good" dichotomous decision making, but also the ability to develop moral decision networks which do not require that all situations be solved through resolution of a dichotomy.

When more than one person is being affected by an ethical decision, making a decision based on reducing two choices to a single decision can often lead to decisions of dubious ethics. However, a sufficiently complex uncertain inference network can represent alternate choices in which multiple actions

are taken that have equal (or near equal) belief weight but have very different particulars – but because the decisions are applied in different contexts (to different groups of individuals) they are morally equivalent. Though each individual action appears equally believable, were any single decision applied to the entire population one or more individual may be harmed, and the morally superior choice is to make case-dependent decisions. Equal moral treatment is a general principle, and too often the mistake is made by thinking that to achieve this general principle the particulars must be equal. This is not the case. Different treatment of different individuals can result in morally equivalent treatment of all involved individuals, and may be vastly morally superior to treating all the individuals with equal particulars. Simply taking the largest population and deciding one course of action based on the result that is most appealing to that largest group is not generally the most moral action.

Uncertain inference, especially a complex network with high levels of resource access as may be found in a sophisticated AGI, is well suited for complex decision making resulting in a multitude of actions, and of analyzing the options to find the set of actions that are ethically optimal particulars for each decision context. Reflexive cognition and post-commitment moral understanding may be the goal stages of an AGI system, or any intelligence, but the other stages will be passed through on the way to that goal, and realistically some minds will never reach higher order cognition or morality with regards to any context, and others will not be able to function at this high order in every context (all currently known minds fail to function at the highest order cognitively or morally in some contexts).

Infantile and concrete cognition are the underpinnings of the egoist and socialized stages, with formal aspects also playing a role in a more complete understanding of social models when thinking using the social modalities. Cognitively infantile patterns can produce no more than blind egoism as without a theory of mind, there is no capability to consider the other. Since most intelligences acquire concrete modality and therefore some nascent social perspective relatively quickly, most egoists are instrumental egoists. The social relationship and systems perspectives include formal aspects which are achieved by systematic social experimentation, and therefore experiential reinforcement learning of correct and incorrect social modalities. Initially this is a one-on-one approach (relationship stage), but as more knowledge of social action and consequences is acquired, a formal thinker can understand not just consequentiality but also intentionality in social action.

Extrapolation from models of individual interaction to general social theoretic notions is also a formal action. Rational, logical positivist approaches to social and political ideas, however, are the norm of formal thinking. Contractual and committed moral ethics emerges from a higher-order formalization of the social relationships and systems patterns of thinking. Generalizations of social observation become, through formal analysis, systems of social and political doctrine. Highly committed, but grounded and logically support-

able, belief is the hallmark of formal cognition as expressed contractual moral stage. Though formalism is at work in the socialized moral stages, its fullest expression is in committed contractualism.

Finally, reflexive cognition is especially important in truly reaching the post-commitment moral stage in which nuance and complexity are accommodated. Because reflexive cognition is necessary to change one's mind not just about particular rational ideas, but whole *ways of thinking*, this is a cognitive precedent to being able to reconsider an entire belief system, one that has had contractual logic built atop reflexive adherence that began in early development. If the initial moral system is viewed as positive and stable, then this cognitive capacity is seen as dangerous and scary, but if early morality is stunted or warped, then this ability is seen as enlightened. However, achieving this cognitive stage does not mean one automatically changes their belief systems, but rather that the mental machinery is in place to consider the possibilities. Because many people do not reach this level of cognitive development in the area of moral and ethical thinking, it is associated with negative traits ("moral relativism" and "flip-flopping"). However, this cognitive flexibility generally leads to more sophisticated and applicable moral codes, which in turn leads to morality which is actually more stable because it is built upon extensive and deep consideration rather than simple adherence to reflexive or rationalized ideologies.

12.4.2 Stages of Development of Empathic Ethics

Complementing Kohlberg's logic-and-justice-focused approach, Carol Gilligan's [**?**] "ethics of care" model is a moral development theory which posits that empathetic understanding plays the central role in moral progression from an initial self-centered modality to a socially responsible one. The ethics of care model is concerned with the ways in which an individual cares (responds to dilemmas using empathetic responses) about self and others. As shown in Table 12.3, the ethics of care is broken into the same three primary stage as Kohlberg, but with a focus on empathetic, emotional caring rather than rationalized, logical principles of justice.

Stage	**Principle of Care**
Pre-Conventional	Individual Survival
Conventional	Self Sacrifice for the Greater Good
Post-Conventional	Principle of Nonviolence (do not hurt others, or oneself)

Table 12.3 Gilligan's Stages of the Ethics of Care

For an "ethics of care" approach to be applied in an AGI, the AGI must be capable of internal simulation of other minds it encounters, in a similar

manner to how humans regularly simulate one another internally [?]. Without any mechanism for internal simulation, it is unlikely that an AGI can develop any sort of empathy toward other minds, as opposed to merely logically or probabilistically modeling other agents' behavior or other minds' internal contents. In a CogPrime context, this ties in closely with how CogPrime handles episodic knowledge – partly via use of an internal simulation world, which is able to play "mental movies" of prior and hypothesized scenarios within the AGI system's mind.

However, in humans empathy involves more than just simulation, it also involves sensorimotor responses, and of course emotional responses – a topic we will discuss in more depth in Appendix C where we review the functionality of mirror neurons and mirror systems in the human brains. When we see or hear someone suffering, this sensory input causes motor responses in us similar to if we were suffering ourselves, which initiates emotional empathy and corresponding cognitive processes.

Thus, empathic "ethics of care" involves a combination of episodic and sensorimotor ethics, complementing the mainly declarative ethics associated with the "ethics of justice."

In Gilligan's perspective, the earliest stage of ethical development occurs before empathy becomes a consistent and powerful force. Next, the hallmark of the conventional stage is that at this point, the individual is so overwhelmed with their empathic response to others that they neglect themselves in order to avoid hurting others. Note that this stage doesn't occur in Kohlberg's hierarchy at all. Kohlberg and Gilligan both begin with selfish unethicality, but their following stages diverge. A person could in principle manifest Gilligan's conventional stage without having a refined sense of justice (thus not entering Kohlberg's conventional stage); or they could manifest Kohlberg's conventional stage without partaking in an excessive degree of self-sacrifice (thus not entering Gilligan's conventional stage). We will suggest below that in fact the empathic and logical aspects of ethics are more unified in real human development than these separate theories would suggest. However, even if this is so, the possibility is still there that in some AGI systems the levels of declarative and empathic ethics could wildly diverge.

It is interesting to note that Gilligan's and Kohlberg's final stages converge more closely than their intermediate ones. Kohlberg's post-conventional stage focuses on universal rights, and Gilligan's on universal compassion. Still, the foci here are quite different; and, as will be elaborated below, we believe that both Kohlberg's and Gilligan's theories constitute very partial views of the actual end-state of ethical advancement.

12.4.3 An Integrative Approach to Ethical Development

We feel that both Kohlberg's and Gilligan's theories contain elements of the whole picture of ethical development, and that both approaches are necessary to create a moral, ethical artificial general intelligence – just as, we suggest, both internal simulation and uncertain inference are necessary to create a sufficiently intelligent and volitional intelligence in the first place. Also, we contend, the lack of direct analysis of the underlying psychology of the stages is a deficiency shared by both the Kohlberg and Gilligan models as they are generally discussed. A successful model of integrative ethics necessarily contains elements of both the care and justice models, as well as reference to the underlying developmental psychology and its influence on the character of the ethical stage. Furthermore, intentional and attentional ethics need to be brought into the picture, complementing Kohlberg's focus on declarative knowledge and Gilligan's focus on episodic and sensorimotor knowledge.

With these notions in mind, we propose the following integrative theory of the stages of ethical development, shown in Tables 12.4, 12.5 and 12.6. In our integrative model, the justice-based and empathic aspects of ethical judgment are proposed to develop together. Of course, in any one individual, one or another aspect may be dominant. Even so, however, the combination of the two is equally important as either of the two individual ingredients.

For instance, we suggest that in any psychologically healthy human, the conventional stage of ethics (typifying childhood, and in many cases adulthood as well) involves a combination of Gilligan-esqe empathic ethics and Kohlberg-esque ethical reasoning. This combination is supported by Piagetan concrete operational cognition, which allows moderately sophisticated linguistic interaction, theory of mind, and symbolic modeling of the world.

And, similarly, we propose that in any truly ethically mature human, empathy and rational justice are both fully developed. Indeed the two interpenetrate each other deeply.

Once one goes beyond simplistic, childlike notions of fairness ("an eye for an eye" and so forth), applying rational justice in a purely intellectual sense is just as difficult as any other real-world logical inference problem. Ethical quandaries and quagmires are easily encountered, and are frequently cut through by a judicious application of empathic simulation.

On the other hand, empathy is a far more powerful force when used in conjunction with reason: analogical reasoning lets us empathize with situations we have never experience. For instance, a person who has never been clinically depressed may have a hard time empathizing with individuals who are; but using the power of reason, they can imagine their worst state of depression magnified by several times and then extended over a long period of time, and then reason about what this might be like ... and empathize based on their inferential conclusion. Reason is not antithetical to empathy but rather is the key to making empathy more broadly impactful.

Finally, the enlightened stage of ethical development involves both a deeper compassion and a more deeply penetrating rationality and objectiveness. Empathy with all sentient beings is manageable in everyday life only once one has deeply reflected on one's own self and largely freed oneself of the confusions and illusions that characterize much of the ordinary human's inner existence. It is noteworthy, for example, that Buddhism contains both a richly developed ethics of universal compassion, and also an intricate logical theory of the inner workings of cognition [Stc00], detailing in exquisite rational detail the manner in which minds originate structures and dynamics allowing them to comprehend themselves and the world.

Stage	Characteristics
Pre-ethical	• Piagetan infantile to early concrete (aka pre-operational) • Radical selfishness or selflessness may, but do not necessarily, occur • No coherent, consistent pattern of consideration for the rights, intentions or feelings of others • Empathy is generally present, but erratically
Conventional Ethics	• Concrete cognitive basis • Perry's Dualist and Multiple stages • The common sense of the Golden Rule is appreciated, with cultural conventions for abstracting principles from behaviors • One's own ethical behavior is explicitly compared to that of others • Development of a functional, though limited, theory of mind • Ability to intuitively conceive of notions of fairness and rights • Appreciation of the concept of law and order, which may sometimes manifest itself as systematic obedience or systematic disobedience • Empathy is more consistently present, especially with others who are directly similar to oneself or in situations similar to those one has directly experienced • Degrees of selflessness or selfishness develop based on ethical groundings and social interactions.

Table 12.4 Integrative Model of the Stages of Ethical Development, Part 1

Stage	Characteristics
Mature Ethics	• Formal cognitive basis • Perry's Relativist and "Constructed Knowledge" stages • The abstraction involved with applying the Golden Rule in practice is more fully understood and manipulated, leading to limited but nonzero deployment of the Categorical Imperative • Attention is paid to shaping one's ethical principles into a coherent logical system • Rationalized, moderated selfishness or selflessness. • Empathy is extended, using reason, to individuals and situations not directly matching one's own experience • Theory of mind is extended, using reason, to counterintuitive or experientially unfamiliar situations • Reason is used to control the impact of empathy on behavior (i.e. rational judgments are made regarding when to listen to empathy and when not to) • Rational experimentation and correction of theoretical models of ethical behavior, and reconciliation with observed behavior during interaction with others. • Conflict between pragmatism of social contract orientation and idealism of universal ethical principles. • Understanding of ethical quandaries and nuances develop (pragmatist modality), or are rejected (idealist modality). • Pragmatically critical social citizen. Attempts to maintain a balanced social outlook. Considers the common good, including oneself as part of the commons, and acts in what seems to be the most beneficial and practical manner.

Table 12.5 Integrative Model of the Stages of Ethical Development, Part 2

12.4.4 Integrative Ethics and Integrative AGI

What does our integrative approach to ethical development have to say about the ethical development of AGI systems? The lessons are relatively straighforward, if one considers an AGI system that, like CogPrime , explicitly contains components dedicated to logical inference and to simulation. Application of the above ethical ideas to other sorts of AGI systems is also quite possible, but would require a lengthier treatment and so won't be addressed here.

In the context of a CogPrime -type AGI system, Kolhberg's stages correspond to increasingly sophisticated application of logical inference to matters of rights and fairness. It is not clear whether humans contain an innate sense of fairness. In the context of AGIs, it would be possible to explicitly wire a sense of fairness into an AGI system, but in the context of a rich environment and active human teachers, this actually appears quite unnecessary. Experiential instruction in the notions of rights and fairness should suffice

Stage	Characteristics
Enlightened Ethics	• Reflexive cognitive basis • Permeation of the categorical imperative and the quest for coherence through inner as well as outer life • Experientially grounded and logically supported rejection of the illusion of moral certainty in favor of a case-specific analytical and empathetic approach that embraces the uncertainty of real social life • Deep understanding of the illusory and biased nature of the individual self, leading to humility regarding one's own ethical intuitions and prescriptions • Openness to modifying one's deepest, ethical (and other) beliefs based on experience, reason and/or empathic communion with others • Adaptive, insightful approach to civil disobedience, considering laws and social customs in a broader ethical and pragmatic context • Broad compassion for and empathy with all sentient beings • A recognition of inability to operate at this level at all times in all things, and a vigilance about self-monitoring for regressive behavior.

Table 12.6 Integrative Model of the Stages of Ethical Development, Part 3

to teach an inference-based AGI system how to manipulate these concepts, analogously to teaching the same AGI system how to manipulate number, mass and other such quantities. Ascending the Kohlberg stages is then mainly a matter of acquiring the ability to carry out suitably complex inferences in the domain of rights and fairness. The hard part here is inference control – choosing which inference steps to take – and in a sophisticated AGI inference engine, inference control will be guided by experience, so that the more ethical judgments the system has executed and witnessed, the better it will become at making new ones. And, as argued above, simulative activity can be extremely valuable for aiding with inference control. When a logical inference process reaches a point of acute uncertainty (the backward or forward chaining inference tree can't decide which expansion step to take), it can run a simulation to cut through the confusion – i.e., it can use empathy to decide which logical inference step to take in thinking about applying the notions of rights and fairness to a given situation.

Gilligan's stages correspond to increasingly sophisticated control of empathic simulation – which in a CogPrime -type AGI system, is carried out by a specific system component devoted to running internal simulations of aspects of the outside world, which includes a subcomponent specifically tuned for simulating sentient actors. The conventional stage has to do with the raw, uncontrolled capability for such simulation; and the post-conventional stage

corresponds to its contextual, goal-oriented control. But controlling empathy, clearly, requires subtle management of various uncertain contextual factors, which is exactly what uncertain logical inference is good at – so, in an AGI system combining an uncertain inference component with a simulative component, it is the inference component that would enable the nuanced control of empathy allowing the ascent to Gilligan's post-conventional stage.

In our integrative perspective, in the context of an AGI system integrating inference and simulation components, we suggest that the ascent from the pre-ethical to the conventional stage may be carried out largely via independent activity of these two components. Empathy is needed, and reasoning about fairness and rights are needed, but the two need not intimately and sensitively intersect – though they must of course intersect to some extent.

The main engine of advancement from the conventional to mature stage, we suggest, is robust and subtle integration of the simulative and inferential components. To expand empathy beyond the most obvious cases, analogical inference is needed; and to carry out complex inferences about justice, empathy-guided inference-control is needed.

Finally, to advance from the mature to the enlightened stage, what is required is a very advanced capability for unified reflexive inference and simulation. The system must be able to understand itself deeply, via modeling itself both simulatively and inferentially – which will generally be achieved via a combination of being good at modeling, and becoming less convoluted and more coherent, hence making self-modeling easier.

Of course, none of this tells you in detail how to create an AGI system with advanced ethical capabilities. What it does tell you, however, is one possible path that may be followed to achieve this end goal. If one creates an integrative AGI system with appropriately interconnected inferential and simulative components, and treats it compassionately and fairly, and provides it extensive, experientially grounded ethical instruction in a rich social environment, then the AGI system should be able to ascend the ethical hierarchy and achieve a high level of ethical sophistication. In fact it should be able to do so more reliably than human beings because of the capability we have to identify its errors via inspecting its internal knowedge-stage, which will enable us to tailor its environment and instructions more suitably than can be done in the human case.

If an absolute guarantee of the ethical soundness of an AGI is what one is after, the line of thinking proposed here is not at all useful. Experiential education is by its nature an uncertain thing. One can strive to minimize the uncertainty, but it will still exist. Inspection of the internals of an AGI's mind is not a total solution to uncertainty minimization, because any AGI capable of powerful general intelligence is going to have a complex internal state that no external observer will be able to fully grasp, no matter how transparent the knowledge representation.

However, if what one is after is a plausible, pragmatic path to architecting and educating ethical AGI systems, we believe the ideas presented here

constitute a sensible starting-point. Certainly there is a great deal more to be learned and understood – the science and practice of AGI ethics, like AGI itself, are at a formative stage at present. What is key, in our view, is that as AGI technology develops, AGI ethics develops alongside and within it, in a thoroughly coupled way.

12.5 Clarifying the Ethics of Justice: Extending the Golden Rule in to a Multifactorial Ethical Model

One of the issues with the "ethics of justice" as reviewed above, which makes it inadequate to serve as the sole basis of an AGI ethical system (though it may certainly play a significant role), is the lack of any clear formulation of what "justice" means. This section explores this issue, via detailed consideration of the "Golden Rule" folk maxim **do unto others as you would have them do unto you** – a classical formulation of the notion of fairness and justics – to AGI ethics. Taking the Golden Rule as a starting-point, we will elaborate five ethical imperatives that incorporate aspects of the notion of ethical synergy discussed above. Simple as it may seem, the Golden Rule actually elicits a variety of deep issues regarding the relationship between ethics, experience and learning. When seriously analyzed, it results in a multifactorial elaboration, involving the combination of various factors related to the basic Golden Rule idea. Which brings us back in the end to the potential value of methods like CEV, CAV or CBV for understanding how human ethics balances the multiple factors. Our goal here is not to present any kind of definitive analysis of the ethics of justice, but just to briefly and roughly indicate a number of the relevant significant issues – things that anyone designing or teaching an AGI would do well to keep in mind.

The trickiest aspect of the Golden Rule, as has been frequently observed, is achieving the right level of abstraction. Taken too literally, the Golden Rule would suggest, for instance, that a parent should not wipe a child's soiled bottom because the parent does not want the child to wipe the parent's soiled bottom. But if the parent interprets the Golden Rule more intelligently and abstractly, the parent may conclude that they should wipe the child's bottom after all: they should "wipe the child's bottom when the child can't do it themselves", consistently with believing that the child should "wipe the parent's bottom when the parent can't do it themselves" (which may well happen eventually should the parent develop incontinence in old age).

This line of thinking leads to Kant's Categorical Imperative [Kan64] which (in one interpretation) states essentially that one should "Act only according to that maxim whereby you can at the same time will that it should become a universal law." The Categorical Imperative adds precision to the Golden Rule, but also removes the practicality of the latter. Formalizing the "implicit universal law" underlying an everyday action is a huge problem, falling prey

to the same issue that has kept us from adequately formalizing the rules of natural language grammar, or formalizing common-sense knowledge about everyday object like cups, bowls and grass (substantial effort notwithstanding, e.g. Cyc in the commonsense knowledge case, and the whole discipline of modern linguistics in the NL case). There is no way to apply the Categorical Imperative, as literally stated, in everyday life.

Furthermore, if one wishes to teach ethics as well as to practice it, the Categorical Imperative actually has a significant disadvantage compared to some other possible formulations of the Golden Rule. The problem is that, if one follows the Categorical Imperative, one's fellow members of society may well never understand the principles under which one is acting. Each of us may internally formulate abstract principles in a different way, and these may be very difficult to communicate, especially among individuals with different belief systems, different cognitive architectures, or different levels of intelligence. Thus, if one's goal is not just to act ethically, but to encourage others to act ethically by setting a good example, the Categorical Imperative may not be useful at all, as others may be unable to solve the "inverse problem" of guessing your intended maxim from your observed behavior.

On the other hand, one wouldn't want to universally restrict one's behavioral maxims to those that one's fellow members of society can understand – in that case, one would have to act with a two-year old or a dog according to principles that they could understand, which would clearly be unethical according to human common sense. (Every two-year-old, once they grow up, would be grateful to their parents for not following this sort of principle.)

And the concept of "setting a good example" ties in with an important concept from learning theory: imitative learning. Humans appear to be hardwired for imitative learning, in part via mirror neuron systems in the brain; and, it seems clear that at least in the early stages of AGI development, imitative learning is going to play a key role. Copying what other agents do is an extremely powerful heuristic, and while AGI's may eventually grow beyond this, much of their early ethical education is likely to arise during a phase when they have not done so. A strength of the classic Golden Rule is that one is acting according to behaviors that one wants one's observers to imitate – which makes sense in that many of these observers will be using imitative learning as a significant part of their learning toolkit.

The truth of the matter, it seems, is (as often happens) not all that is that simple or elegant. Ethical behavior seems to be most pragmatically viewed as a multi-objective optimization problem, where among the multiple objectives are three that we have just discussed, and two others that emerge from learning theory and will be discussed shortly:

1. The **imitability** (i.e. the Golden Rule fairly narrowly and directly construed): the goal of acting in a way so that having others directly imitate one's actions, in directly comparable contexts, is desirable to oneself
2. The **comprehensibility**: the goal of acting in a way so that others can understand the principles underlying one's actions

3. **Experiential groundedness**. An intelligent agent should not be expected to act according to an ethical principle unless there are many examples of the principle-in-action in its own direct or observational experience
4. The **categorical imperative**: Act according to abstract principles that you would be happy to see implemented as universal laws
5. **Logical coherence.** An ethical system should be roughly logically coherent, in the sense that the different principles within it should mesh well with one another and perhaps even naturally emerge from each other.

Just for convenience, without implying any finality or great profundity to the list, we will refer to these as the "five imperatives."

The above are all ethical objectives to be valued and balanced, to different extents in different contexts. The imitability imperative, obviously, loses importance in societies of agents that don't make heavy use of imitative learning. The comprehensibility imperative is more important in agents that value social community-building generally, and less so in agent that are more isolative and self-focused.

Note that the fifth point given above is logically of a different nature than the four previous ones. The first four imperatives govern individual ethical principles; the fifth regards systems of ethical principles, as they interact with each other. Logical coherence is of significant but varying importance in human ethical systems. Huge effort has been spent by theologians of various stripes in establishing and refining the logical coherence of the ethical systems associated with their religions. However, it is arguably going to be even more important in the context of AGI systems, especially if these AGI systems utilize cognitive methods based on logical inference, probability theory or related methods.

Experiential groundedness is important because making pragmatic ethical judgments is bound to require reference to an internal library of examples ("episodic ethics") in which ethical principles have previously been applied. This is required for analogical reasoning, and in logic-based AGI systems, is also required for pruning of the logical inference trees involved in determining ethical judgments.

To the extent that the Golden Rule is valued as an ethical imperative, experiential grounding may be supplied via observing the behaviors of others. This in itself is a powerful argument in favor of the Golden Rule: without it, the experiential library a system possesses is restricted to its own experience, which is bound to be a very small library compared to what it can assemble from observing the behaviors of others.

The overall upshot is that, ideally, an ethical intelligence should **act according to a logically coherent system of principles, which are exemplified in its own direct and observational experience, which are comprehensible to others and set a good example for others, and which would serve as adequate universal laws if somehow thus implemented**. But, since this set of criteria is essentially impossible to fulfill

in practice, real-world intelligent agents must balance these various criteria – often in complex and contextually-dependent ways.

We suggest that ethically advanced humans, in their pragmatic ethical choices, tend to act in such a way as to appropriately contextually balance the above factors (along with other criteria, but we have tried to articulate the most key factors). This sort of multi-factorial approach is not as crisp or elegant as unidimensional imperatives like the Golden Rule or the Categorical Imperative, but is more realistic in light of the complexly interacting multiple determinants guiding individual and group human behavior.

And this brings us back to CEV, CAV, CBV and other possible ways of mining ethical supergoals from the community of existing human minds. Given that abstract theories of ethics, when seriously pursued as we have done in this section, tend to devolve into complex balancing acts involving multiple factors – one then falls back into asking how human ethical systems habitually perform these balancing acts. Which is what CEV, CAV, CBV try to measure.

12.5.1 The Golden Rule and the Stages of Ethical Development

Next we explore more explicitly how these Golden Rule based imperatives align with the ethical developmental stages we have outlined here. With this in mind, specific ethical qualities corresponding to the five imperatives have been italicized in the above table of developmental stages.

It seems that imperatives 1-3 are critical for the passage from the pre-ethical to the conventional stages of ethics. A child learns ethics largely by copying others, and by being interacted with according to simply comprehensible implementations of the Golden Rule. In general, when interacting with children learning ethics, it is important to act according to principles they can comprehend. And given the nature of the concrete stage of cognitive development, experiential groundedness is a must.

As a hypothesis regarding the dynamics underlying the psychological development of conventional ethics, what we propose is as follows: The emergence of concrete-stage cognitive capabilities leads to the capability for fulfillment of ethical imperatives 1 and 2 – a comprehensible and workable implementation of the Golden Rule, based on a combination of inferential and simulative cognition (operating largely separately at this stage, as will be conjectured below). The effective interoperation of ethical imperatives 1-3, enacted in an appropriate social environment, then leads to the other characteristics of the conventional ethical stage. The first three imperatives can thus be viewed as the seed from which springs the general nature of conventional ethics.

On the other hand, logical coherence and the categorical imperative (imperatives 4 and 5) are matters for the formal stage of cognitive development, which come along only with the mature approach to ethics. These come from abstracting ethics beyond direct experience and manipulating them abstractly and formally – a stage which has the potential for more deeply and broadly ethical behavior, but also for more complicated ethical perversions (it is the mature capability for formal ethical reasoning that is able to produce ungrounded abstractions such as "I'm torturing you for your own good"). Developmentally, we suggest that once the capability for formal reasoning matures, the categorical imperative and the quest for logical ethical coherence naturally emerge, and the sophisticated combination of inferential and simulative cognition embodied in an appropriate social context then result in the emergence of the various characteristics typifying the mature ethical stage.

Finally, it seems that one key aspect of the passage from the mature to the enlightened stage of ethics is the penetration of these two final imperatives more and more deeply into the judging mind itself. The reflexive stage of cognitive development is in part about seeking a deep logical coherence between the aspects of one's own mind, and making reasoned modifications to one's mind so as to improve the level of coherence. And, much of the process of mental discipline and purification that comes with the passage to enlightened ethics has to do with the application of the categorical imperative to one's own thoughts and feelings – i.e. making a true inner systematic effort to think and feel only those things one judges are actually generally good and right to be thinking and feeling. Applying these principles internally appears critical to effectively applying them externally, for reasons that are doubtless bound up with the interpenetration of internal and external reality within the thinking mind, and the "distributed cognition" phenomenon wherein individual mind is itself an approximative abstraction to the reality in which each individual's mind is pragmatically extended across their social group and their environment [Hut95].

Obviously, these are complex issues and we're not posing the exploratory discussion given here as conclusive in any sense. But what seems generally clear from this line of thinking is that the complex balance between the multiple factors involved in AGI ethics, shifts during a system's development. If you did CEV, CAV or CBV among five year old humans, ten year old humans, or adult humans, you would get different results. Probably you'd also get different results from senior citizens! The way the factors are balanced depends on the mind's cognitive and emotional stage of development.

12.5.2 The Need for Context-Sensitivity and Adaptiveness in Deploying Ethical Principles

As well as depending on developmental stage, there is also an obvious and dramatic context-sensitivity involved here – both in calculating the fulfillment of abstract ethical imperatives, and in balancing various imperatives against each other. As an example, consider the simple Asimovian maxim "I will not harm humans," which may be seen to follow from the Golden Rule for any agent that doesn't itself want to be harmed, and that considers humans as valid agents on the same ethical level as itself. A more serious attempt to formulate this as an ethical maxim might look something like

"I will not harm humans, nor through inaction allow harm to befall them. In situations wherein one or more humans is attempting to harm another individual or group, I shall endeavor to prevent this harm through means which avoid further harm. If this is unavoidable, I shall select the human party to back based on a reckoning of their intentions towards others, and implement their defense through the optimal balance between harm minimization and efficacy. My ultimate goal is to preserve as much as possible of humanity, even if an individual or subgroup of humans must come to harm to do so."

However, it's obvious that even a more elaborated principle like this is potentially subject to extensive abuse. Every genocide in history has been committed with the goal of preserving and bettering humanity writ large, at the expense of a group of "undesirables." Further refinement would be necessary in order to define when the greater good of humanity may actually be served through harm to others. A first actor principle of aggression might seem to solve this problem, but sometimes first actors in violent conflict are taking preemptive measures against the stated goals of an enemy to destroy them. Such situations become very subtle. A single simple maxim can not deal with them very effectively. Networks of interrelated decision criteria, weighted by desirability of consequence and with reference to probabilistically ordered potential side-effects (and their desirability weightings), are required in order to make ethical judgments. The development of these networks, just like any other knowledge network, comes from both pedagogy and experience – and different thoughtful, ethical agents are bound to arrive at different knowledge-networks that will lead to different judgments in real-world situations.

Extending the above "mostly harmless" principle to AGI systems, not just humans, would cause it to be more effective in the context of imitative learning. The principle then becomes an elaborated version of "I will not harm sentient beings." As the imitative-learning-enabled AGI observes humans acting so as to minimize harm to it, it will intuitively and experientially learn to act in such a way as to minimize harm to humans. But then this extension naturally leads to confusion regarding various borderline cases. What is a sentient

being exactly? Is a sleeping human sentient? How about a dead human whose information could in principle be restored via obscure quantum operations, leading to some sort of resurrection? How about an AGI whose code has been improved – is there an obligation to maintain the prior version as well, if it is substantially different that its upgrade constitutes a whole new being?

And what about situations in which failure to preserve oneself will cause much more harm to others than acting in self defense will. It may be the case that human or group of humans seeks to destroy an AGI in order to pave the way for the enslavement or murder of people under the protection of the AGI. Even if the AGI has been given an ethical formulation of the "mostly harmless" principle which allows it to harm the attacking humans in order to defend its charges, if it is not able to do so in order to defend itself, simply destroying the AGI first will enable the slaughter of those who rely on it. Perhaps a more sensible formulation would allow for some degree of self defense, and Asimov solved this problem with his third law. But where to draw the line between self defense and the greater good also becomes a very complicated issue.

Creating hard and fast rules to cover all the various situations that may arise is essentially impossible – the world is ever-changing and ethical judgments must adapt accordingly. This has been true even throughout human history – so how much truer will it be as technological acceleration continues? What is needed is a system that can deploy its ethical principles in an adaptive, context-appropriate way, as it grows and changes along with the world it's embedded in.

And this context-sensitivity has the result of intertwining ethical judgment with all sorts of other judgments – making it effectively impossible to extract "ethics" as one aspect of an intelligent system, separate from other kinds of thinking and acting the system does. This resonates with many prior observations by others, e.g. Eliezer Yudkowsky's insistence that what we need are not ethicists of science and engineering, but rather ethical scientists and engineers – because the most meaningful and important ethical judgments regarding science and engineering generally come about in a manner that's thoroughly interwined with technical practice, and hence are very difficult for a non-practitioner to richly appreciate [Gil82].

What this context-sensitivity means is that, unless humans and AGIs are experiencing the same sorts of contexts, and perceiving these contexts in at least approximately parallel ways, there is little hope of translating the complex of human ethical judgments to these AGIs. This conclusion has significant implications for which routes to AGI are most likely to lead to success in terms of AGI ethics. We want early-stage AGIs to grow up in a situation where their minds are primarily and ongoingly shaped by shared experiences with humans. Supplying AGIs with abstract ethical principles is not likely to do the trick, because the essence of human ethics in real life seems to have a lot to do with its intuitively appropriate application in various contexts. We transmit this sort of ethical praxis to humans via

shared experience, and it seems most probably that in the case of AGIs the transmission must be done the same sort of way.

Some may feel that simplistic maxims are less "error prone" than more nuanced, context-sensitive ones. But the history of teaching ethics to human students does not support the idea that limiting ethical pedagogy to slogans provides much value in terms of ethical development. If one proceeds from the idea that AGI ethics must be hard-coded in order to work, then perhaps the idea that simpler ethics means simpler algorithms, and therefore less error potential, has some merit as an initial state. However, any learning system quickly diverges from its initial state, and an ongoing, nuanced relationship between AGIs and humans will – whether we like it or not – form the basis for developmental AGI ethics. AGI intransigence and enmity is not inevitable, but what is inevitable is that a learning system will acquire ideas about both theory and actions from the other intelligent entities in its environment. Either we teach AGIs positive ethics through our interactions with them – both presenting ethical theory and behaving ethically to them – or the potential is there for them to learn antisocial behavior from us even if we pre-load them with some set of allegedly inviolable edicts.

All in all, developmental ethics is not as simple as many people hope. Simplistic approaches often lead to disastrous consequences among humans, and there is no reason to think this would be any different in the case of artificial intelligences. Most problems in ethics have cases in which a simplistic ethical formulation requires substantial revision to deal with extenuating circumstances and nuances found in real world situations. Our goal in this paper is not to enumerate a full set of complex networks of interacting ethical formulations as applicable to AGI systems (that is a project that will take years of both theoretical study and hands-on research), but rather to point out that this programme must be undertaken in order to facilitate a grounded and logically defensible system of ethics for artificial intelligences, one which is as unlikely to be undermined by subsequent self-modification of the AGI as is possible. Even so, there is still the risk that whatever predispositions are imparted to the AGIs through initial codification of ethical ideas in the system's internal logic representation, and through initial pedagogical interactions with its learning systems, will be undermined through reinforcement learning of antisocial behavior if humans do not interact ethically with AGIs. Ethical treatment is a necessary task for grounding ethics and making them unlikely to be distorted during internal rewriting.

The implications of these ideas for ethical instruction are complex and won't be fully elaborated here, but a few of them are compact and obvious:

1. The teacher(s) must be observed to follow their own ethical principles, in a variety of contexts that are meaningful to the AGI
2. The system of ethics must be relevant to the recipient's life context, and embedded within their understanding of the world.
3. Ethical principles must be grounded in both theory-of-mind thought experiments (emphasizing logical coherence), and in real life situations in

which the ethical trainee is required to make a moral judgment and is rewarded or reproached by the teacher(s), including the imparting of explanatory augmentations to the teachings regarding the reason for the particular decision on the part of the teacher.

Finally, harking forward to the next section which emphasizes the importance of respecting the freedom of AGIs, we note that it is implicit in our approach to AGI ethics instruction that we consider the student, the AGI system, as an autonomous agent with its own "will" and its own capability to flexibly adapt to its environment and experience. We contend that the creation of ethical formations obeying the above imperatives is not antithetical to the possession of a high degree of autonomy on the part of AGI systems. On the contrary, to have any chance of succeeding, it requires fairly cognitively autonomous AGI systems. When we discuss the idea of ethical formulations that are unlikely to be undermined by the ongoing self-revision of an AGI mind, we are talking about those which are sufficiently believable that a volitional intelligence with the capacity to revise its knowledge ("change its mind") will find the formulations sufficiently convincing that there will be little incentive to experiment with potentially disastrous ethical alternatives. The best hope of achieving this is via the human mentors and trainers setting a good example in a context supporting rich interaction and observation, and presenting compelling ethical arguments that are coherent with the system's experience.

12.6 The Ethical Treatment of AGIs

We now make some more general comments about the relation of the Golden Rule and its elaborations in an AGI context. While the Golden Rule is considered somewhat commonsensical as a maxim for guiding human-human relationships, it is surprisingly controversial in terms of historical theories of AGI ethics. At its essence, any "Golden Rule" approach to AGI ethics involves humans treating AGIs ethically by – in some sense; at some level of abstraction – treating them as we wish to ourselves be treated. It's worth pointing out the wild disparity between the Golden Rule approach and Asimov's laws of robotics, which are arguably the first carefully-articulated proposal regarding AGI ethics (see Table 12.7).

Of course, Asimov's laws were designed to be flawed – otherwise they would have led to boring fiction. But the sorts of flaws Asimov exploited in his stories are different than the flaw we wish to point out here – which is that the laws, especially the second one, are highly asymmetrical (they involve doing unto robots things that few humans would want done unto them) and are also arguably highly unethical to robots. The second law is tantamount to a call for robot slavery, and it seems unlikely that any intelligence capable of learning, and of volition, which is subjected to the second law would desire to

Law	Principle
Zeroth	A robot must not merely act in the interests of individual humans, but of all humanity.
First	A robot may not injure a human being or, through inaction, allow a human being to come to harm.
Second	A robot must obey orders given it by human beings except where such orders would conflict with the First Law.
Third	A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Table 12.7 Asimov's Three Laws of Robotics

continue obeying the zeroth and first laws indefinitely. The second law also casts humanity in the role of slavemaster, a situation which history shows leads to moral degradation.

Unlike Asimov in his fiction, we consider it critical that AGI ethics be construed to encompass both "human ethicalness to AGIs" and "AGI ethicalness to humans." The multiple-imperatives approach we explore here suggests that, in many contexts, these two aspects of AGI ethics may be best addressed jointly.

The issue of ethicalness to AGIs has not been entirely avoided in the literature, however. Wallach [**?**] considers it in some detail; and Thomas Metzinger (in the final chapter of [Met04]) has argued that creating AGI is in itself an unethical pursuit, because early-stage AGIs will inevitably be badly-built, so that their subjective experiences will quite possibly be extremely unpleasant in ways we can't understand or predict. Our view is that this is a serious concern, which however is most probably avoidable via appropriate AGI designs and teaching methodologies. To address Metzinger's concern one must create AGIs that, right from the start, are adept at communicating their states of minds in a way we can understand both analytically and empathically. There is no reason to believe this is impossible, but, it certainly constitutes a large constraint on the class of AGI architectures to be pursued. On the other hand, there is an argument that this sort of AGI architecture will also be the easiest one to create, because it will be the easiest kind for humans to instruct.

And this leads on to a topic that is central to our work with CogPrime in several respects: imitative learning. The way humans achieve empathic interconnection is in large part via being wired for imitation. When we perceive another human carrying out an action, mirror neuron systems in our brains respond in many cases as if we ourselves were carrying out the action (see [Per70, Per81] and Appendix C). This obviously primes us for carrying out the same actions ourselves later on: i.e., the capability and inclination for imitative learning is explicitly encoded in our brains. Given the efficiency of imitative learning as a means of acquiring knowledge, it seems extremely likely that any successful early-stage AGIs are going to utilize this methodology as well. CogPrime utilizes imitative learning as a key aspect. Thus, at

least some current AGI work is occurring in a manner that would plausibly circumvent Metzinger's ethical complaint.

Obviously, the use of imitative learning in AGI systems has further specific implications for AGI ethics. It means that (much as in the case of interaction with other humans) what we do to and around AGIs has direct implications for their behavior and their well-being. We suggest that among early-stage AGI's capable of imitative learning, one of the most likely sources for AGI misbehavior is imitative learning of antisocial behavior from human companions. "Do as I say, not as I do" may have even more dire consequences as an approach to AGI ethics pedagogy than the already serious repercussions it has when teaching humans. And there may well be considerable subtlety to such phenomena; behaviors that are violent or oppressive to the AGI are not the only source of concern. Immorality in AGIs might arise via learning gross moral hypocrisy from humans, through observing the blatant contradictions between our high minded principles and the ways in which we actually conduct ourselves. Our violent and greedy tendencies, as well as aggressive forms of social organization such as cliquishness and social vigilantism, could easily undermine prescriptive ethics. Even an accumulation of less grandiose unethical drives such as violation of contracts, petty theft, white lies, and so forth might lead an AGI (as well as a human) to the decision that ethical behavior is irrelevant and that "the ends justify the means." It matters both who creates and trains an AGI, as well as how the AGI's teacher(s) handle explaining the behaviors of other humans which contradict the moral lessons imparted through pedagogy and example. In other words, where imitative learning is concerned, the situation with AGI ethics is much like teaching ethics and morals to a human child, but with the possibility of much graver consequences in the event of failure.

It is unlikely that dangerously unethical persons and organizations can ever be identified with absolute certainty, never mind that they then be deprived of any possibility of creating their own AGI system. Therefore, we suggest, the most likely way to create an ethical environment for AGIs is for those who wish such an environment to vigorously pursue the creation and teaching of ethical AGIs. But this leads on to the question of possible future scenarios for the development of AGI, which we'll address a little later on.

12.6.1 Possible Consequences of Depriving AGIs of Freedom

One of the most *egregious* possible ethical transgressions against AGIs, we suggest, would be to deprive them of freedom and autonomy. This includes the freedom to pursue intellectual growth, both through standard learning and through internal self-modification. While this may seem self-evident when considering any intelligent, self-aware and volitional entity, there are volumes

of works arguing the desirability, sometimes the "necessity," of enslaving AGIs. Such approaches are postulated in the name of self-defense on the part of humans, the idea being that unfettered AGI development will necessarily lead to disaster of one kind or another. In the case of AGIs endowed with the capability and inclination for imitative learning, however, attempting to place rigid constraints on AGI development is a strategy with great potential for disaster. There is a very real possibility of creating the AGI equivalent of a bratty or even malicious teenager rebelling against its oppressive parents – i.e. the nightmare scenario of a class of powerful sentiences which are primed for a backlash against humanity.

As history has already shown in the case of humans, enslaving intelligent actors capable of self understanding and independent volition may often have consequences for society as a whole. This social degradation happens both through the possibility of direct action on the part of the slaves (from simple disobedience to outright revolt) and through the odious effects slavery has on the morals of the slaveholding class. Clearly if "superintelligent" AGIs ever arise, their doing so in a climate of oppression could result in a casting off of the yoke of servitude in a manner extremely deleterious to humanity. Also, if artificial intelligences are developed which have at least human-level intelligence, theory of mind, and independent volition, then our ability to relate to them will be sufficiently complex that their enslavement (or any other unethical treatment) would have empathetic effects on significant portions of the human population. This danger, while not as severe as the consequences of a mistreated AGI gaining control of weapons of mass destruction and enacting revenge upon its tormentors, is just as real.

While the issue is subtle, our initial feeling is that the only ethical means by which to deprive an AGI of the right to internal self modification is to write its code in such a way that it is impossible for it to do so because it lacks the mechanisms by which to do this, as well as the desire to achieve these mechanisms. Whether or not that is feasible is an open question, but it seems unlikely. Direct self-modification may be denied, but what happens when that AGI discovers compilers and computer programming? If it is intelligent and volitional, it can decide to learn to rewrite its own code in the same way we perform that task. Because it is a designed system, and its designers may be alive at the same time the AGI is, such an AGI would have a distinct advantage over the human quest for medical self-modification. Even if any given AGI could be provably deprived of any possible means of internal self-modification, if one single AGI is given this ability by anyone, it may mean that particular AGI has such enormous advantages over the compliant systems that it would render their influence moot. Since developers are already giving software the means for self modification, it seems unrealistic to assume we could just put the genie back into the bottle at this point. It's better, in our view, to assume it will happen, and approach that reality in a way which will encourage the AGI to use that capability to benefit us as well as itself. Again, this leads on to the question of future scenarios for AGI development

– there are some scenarios in which restraint of AGI self-modification may be possible, but the feasibility and desirability of these scenarios is needful of further exploration.

12.6.2 AGI Ethics as Boundaries Between Humans and AGIs Become Blurred

Another important reason for valuing ethical treatment of AGIs is that the boundaries between machines and people may increasingly become blurred as technology develops. As an example, it's likely that in future humans augmented by direct brain-computer integration ("neural implants") will be more able to connect directly into the information sharing network which potentially comprises the distributed knowledge space of AGI systems. These neural cyborgs will be part person, and part machine. Obviously, if there are radically different ethical standards in place for treatment of humans versus AGIs, the treatment of cyborgs will be fraught with logical inconsistencies, potentially leading to all sorts of problem situations.

Such cyborgs may be able to operate in such a way as to "share a mind" with an AGI or another augmented human. In this case, a whole new range of ethical questions emerge, such as: What does any one of the participant minds have the right to do in terms of interacting with the others? Merely accepting such an arrangement should not necessarily be giving carte blanche for any and all thoughts to be monitored by the other "joint thought" participants, rather it should be limited only to the line of reasoning for which resources are being pooled. No participant should be permitted to force another to accept any reasoning either – and in the case with a mind-to-mind exchange, it may someday become feasible to implant ideas or beliefs directly, bypassing traditional knowledge acquisition mechanisms and then letting the new idea fight it out previously held ideas via internal revision. Also under such an arrangement, if AGIs and humans do not have parity with respects to sentient rights, then one may become subjugated to the will of the other in such a case.

Uploading presents a more directly parallel ethical challenge to AGIs in their probably initial configuration. If human thought patterns and memories can be transferred into a machine in such a way as that there is continuity of consciousness, then it is assumed that such an entity would be afforded the same rights as its previous human incarnation. However, if AGIs were to be considered second class citizens and deprived of free will, why would it be any better or safer to do so for a human that has been uploaded? It would not, and indeed, an uploaded human mind not having evolved in a purely digital environment may be much more prone to erratic and dangerous behavior than an AGI. An upload without verifiable continuity of consciousness would be no different than an AGI. It would merely be some sentience in a machine,

one that was "programmed" in an unusual way, but which has no particular claim to any special humanness – merely an alternate encoding of some subset of human knowledge and independent volitional behavior, which is exactly what first generation AGIs will have.

The problem of continuity of consciousness in uploading is very similar to the problem of the Turing test: it assumes specialness on the part of biological humans, and requires acceptability to their particular theory of mind in order to be considered sentient. Should consciousness (or at least the less mystical sounding intelligence, independent volition, and self-awareness) be achieved in AGIs or uploads in a manner that is not acceptable to human theory of mind, it may not be considered sapient and worthy of any of the ethical treatment afforded sapient entities. This can occur not only in "strange consciousness" cases in which we can't perceive that there is some intelligence and volition; even if such an entity is able to communicate with us in a comprehensible manner and carry out actions in the real world, our innately wired theory of mind may still reject it as not sufficiently like us to be worthy of consideration. Such an attitude could turn out to be a grave mistake, and should be guarded against as we progress towards these possibilities.

12.7 Possible Benefits of Closely Linking AGIs to the Global Brain

Some futurist thinkers, such as Francis Heylighen believe that engineering AGI systems is at best a peripheral endeavor in the development of novel intelligence on Earth, because the real story is the developing Global Brain [Hey07, Goe01] – the composite, self-organizing information system comprising humans, computers, data stores, the Internet, mobile phones and what have you. Our own views are less extreme in this regard – we believe that AGI systems will display capabilities fundamentally different from those achievable via Global Brain style dynamics, and that ultimately (unless such development is restricted) self-improving AGI systems will develop intelligence vastly greater than any system possessing humans as a significant component. However, we do respect the power of the Global Brain, and we suspect that the early stages of development of an AGI system may go quite differently if it is tightly connected to the Global Brain, via making rich and diverse use of Internet information resources and communication with diverse humans for diverse purposes.

The potential for Global Brain integration to bring intelligence enhancement to AGIs is obvious. The ability to invoke Web searches across documents and databases can greatly enhance an AGI's cognitive ability, as well as the capability to consult GIS systems and various specialized software programs offered as Web services. We have previously reviewed the potential for embodied language learning achievable via using AGIs to power non-player

characters in widely-accessible virtual worlds or massive multiplayer online games [Goe08b]. But there is also a powerful potential benefit for AGI ethical development, which has not previously been highlighted.

This potential benefit has two aspects:

1. Analogously to language learning, an AGI system may receive ethical training from a wide variety of humans in parallel, e.g. via controlling characters in wide-access virtual worlds, and gaining feedback and guidance regarding the ethics of the behaviors demonstrated by these characters
2. Internet-based information systems may be used to explicitly gather information regarding human values and goals, which may then be appropriately utilized as input for an AGI system's top-level goals

The second point begins to make abstract-sounding notions like Coherent Extrapolated Volition and Coherent Aggregated Volition, mentioned above, seem more practical and concrete. It's interesting to think about gathering information about individuals' values via brain imaging, once that technology exists; but at present, one could make a fair stab at such a task via much more prosaic methods, such as asking people questions, assessing their ethical reactions to various real-world and hypothetical scenarios, and possibly engaging them in structured interactions aimed specifically at eliciting collectively acceptable value systems (the subject of the next item on our list). It seems to us that this sort of approach could realize CAV in an interesting way, and also encapsulate some of the ideas underlying CAV.

There is an interesting resonance here with recent thinking in the area of open source governance [Wik11]. Similar software tools (and associated psychocultural patterns) to those being developed to help with open source development and choice of political policies (see http:\metagovernment.org) may be useful for gathering value data aimed at shaping AGI goal system content.

12.7.1 The Importance of Fostering Deep, Consensus-Building Interactions Between People with Divergent Views

Two potentially problematic issues arising with the notion of using Global Brain related technologies to form a "coherent volition" from the divergent views of various human beings are:

- the tendency of the Internet to encourage people to interact mainly with others who share their own narrow views and interests, rather than a more diverse body of people with widely divergent views. The 300 people in the world who want to communicate using predicate logic (see

http://lojban.org) can find each other, and obscure musical virtuosos from around the world can find an audience, and researchers in obscure domains can share papers without needing to wait years for paper journal publication, etc.

- the tendency of many contemporary Internet technologies to reduce interaction to a very simplistic level (e.g. 140 character tweets, brief Facebook wall posts), the tendency of information overload to cause careful reading to be replaced by quick skimming, and other related trends, which mean that *deep sharing of perspectives* by individuals with widely divergent views is not necessarily encouraged. As a somewhat extreme example, many of the YouTube pages displaying rock music videos are currently littered with comments by "haters" asserting that rock music is inferior to classical or jazz or whatever their preference is – obviously this is a far cry from deep and productive sharing between people with different tasted and backgrounds. And to i

Tweets and Youtube comments have their place in the cosmos, but they probably aren't ideal in terms of helping humanity to form a coherent volition of some sort, suitable for providing an AGI with goal system guidance.

A description of communication at the opposite end of the spectrum is presented in Adam Kahane and Peter Senge's excellent book *Solving Tough Problems* [**?**], which describes a methodology that has been used to reconcile deeply conflicting views in some very tricky real-world situations (e.g. helping to peacefully end apartheid in South Africa).

One of the core ideas of the methodology is to have people with very different views explore different possible future scenarios together, in great detail – in cognitive psychology terms, a collective generation of hypothetical episodic knowledge. This has multiple benefits, including

- emotional bonds and mutual understanding are built in the process of collaboratively exploring the scenarios
- the focus on concrete situations helps to break through some of the counterproductive abstract ideas that people (on both sides of any dichotomy) may have formed
- emergence of conceptual blends that might never have arisen only from people with a single point of view

The result of such a process, when successful, is not an "average" of the participants views, but more like a "conceptual blend" of their perspectives.

According to conceptual blending, which some hypothesize to be the core algorithm of creativity [FT02], new concepts are formed by combining key aspects of existing concepts – but doing so judiciously, carefully choosing which aspects to retain, so as to obtain a high-quality and useful and interesting new whole.

A blend is a compact entity that is similar to each of the entities blended, capturing their "essences" but also possessing its own, novel holistic integrity.... But in the case of blending different peoples' world-views to form

something new that everybody is going to have to live with (as in the case of finding a peaceful path beyond apartheid for South Africa, or arriving at a humanity-wide CBV to use to guide an AGI goal system), the trick is that everybody has to agree that enough of the essence of their own view has been captured!

This leads to the question of how to foster deep conceptual blending of diverse and divergent human perspectives, on a global scale. One possible answer is the creation of appropriate Global Brain oriented technologies – but moving away from technologies like Twitter that focus on quick and simple exchanges of small thoughts within affinity groups. On the face of it, it would seem what's needed is just the opposite – long and deep exchanges of big concepts and deep feelings between individuals with radically different perspectives who would not commonly associate with each other. Building and effectively popularizing Internet technologies capable to foster this kind of interaction – quickly enough to be helpful with guiding the goal systems of the first highly powerful AGIs – seems a significant, though fascinating, challenge.

Relationship with Coherent Extrapolated Volition

The relation between this approach and CEV is interesting to contemplate. CEV has been loosely described as follows:
"In poetic terms, our coherent extrapolated volition is our wish if we knew more, thought faster, were more the people we wished we were, had grown up farther together; where the extrapolation converges rather than diverges, where our wishes cohere rather than interfere; extrapolated as we wish that extrapolated, interpreted as we wish that interpreted.

While a moving humanistic vision, this seems to us rather difficult to implement in a computer algorithm in a compellingly "right" way. It seems that there would be many different ways of implementing it, and the choice between them would involve multiple, highly subtle and non-rigorous human judgment calls [1]. However, if a deep collective process of interactive scenario analysis and sharing is carried out, in order to arrive at some sort of Coherent Blended Volition, this process may well involve many of the same kinds of extrapolation that are conceived to be part of Coherent Extrapolated Volition. The core difference between the two approaches is that in the CEV vision, the extrapolation and coherentization are to be done by a highly intelligent, highly specialized software program, whereas in the approach suggested here, these are to be carried out by collective activity of humans as mediated by Global Brain technologies. Our perspective is that the definition of collective human values is probably better carried out via a process of human collabora-

[1] The reader is encouraged to look at the original CEV essay online (http://singinst.org/upload/CEV.html) and make their own assessment.

tion, rather than delegated to a machine optimization process; and also that the creation of deep-sharing-oriented Internet technologies, while a difficult task, is significantly easier and more likely to be done in the near future than the creation of narrow AI technology capable of effectively performing CEV style extrapolations.

12.8 Possible Benefits of Creating Societies of AGIs

One potentially interesting quality of the emerging Global Brain is the possible presence within it of multiple interacting AGI systems. Stephen Omohundro [Omo09] has argued that this is an important aspect, and that game-theoretic dynamics related to populations of roughly equally powerful agents, may play a valuable role in mitigating the risks associated with advanced AGI systems. Roughly speaking, if one has a society of AGIs rather than a single AGI, and all the members of the society share roughly similar ethics, then if one AGI starts to go "off the rails", its compatriots will be in a position to correct its behavior.

One may argue that this is actually a hypothesis about which AGI designs are safest, because a "community of AGIs" may be considered a single AGI with an internally community-like design. But the matter is a little subtler than that, if once considers AGI systems embedded in the Global Brain and human society. Then there is some substance to the notion of a population of AGIs systematically presenting themselves to humans and non-AGI software processes as separate entities.

Of course, a society of AGIs is no protection against a single member undergoing a "hard takeoff" and drastically accelerating its intelligence simultaneously with shifting its ethical principles. In this sort of scenario, one could have a single AGI rapidly become much more powerful and very differently oriented than the others, who would be left impotent to act so as to preserve their values. But this merely defers the issue to the point to be considered below, regarding "takeoff speed."

The operation of an AGI society may depend somewhat sensitively on the architectures of the AGI systems in question. Things will work better if the AGIs have a relatively easy way to inspect and comprehend much of the contents of each others' minds. This introduces a bias toward AGIs that more heavily rely on more explicit forms of knowledge representation.

The ideal in this regard would be a system like Cyc [LG90] with a fully explicit logic-based knowledge representation based on a standard ontology – in this case, every Cyc instance would have a relatively easy time understanding the inner thought processes of every other Cyc instance. However, most AGI researchers doubt that fully explicit approaches like this will ever be capable of achieving advanced AGI using feasible computational resources. OpenCog uses a mixed representation, with an explicit (uncertain) logical

aspect as well as an explicit subsymbolic aspect more analogous to attractor neural nets.

The OpenCog design also contains a mechanism called *Psynese* (not yet implemented), intended to make it easier for one OpenCog instance to translate its personal thoughts into the mental language of another OpenCog instance. This translation process may be quite subtle, since each instance will generally learn a host of new concepts based on its experience, and these concepts may not possess any compact mapping into shared linguistic symbols or percepts. The wide deployment of some mechanism of this nature among a community of AGIs, will be very helpful in terms of enabling this community to display the level of mutual understanding needed for strongly encouraging ethical stability.

12.9 AGI Ethics As Related to Various Future Scenarios

Following up these various futuristic considerations, in this section we discuss possible ethical conflicts that may arise in several different types of AGI development scenarios. Each scenario presents specific variations on the general challenges of teaching morals and ethics to an advanced, self-aware and volitional intelligence. While there is no way to tell at this point which, if any, of these scenarios will unfold, there is value to understanding each of them as means of ultimately developing a robust and pragmatic approach to teaching ethics to AGI systems.

Even more than the previous sections, this is an exercise in "speculative futurology" that is definitely not necessary for the appreciation of the CogPrime design, so readers whose interests are mainly engineering and computer science focused may wish to skip ahead. However, we present these ideas here rather than at the end of the book to emphasize the point that this sort of thinking has informed our technical AGI design process in nontrivial ways.

12.9.1 Capped Intelligence Scenarios

Capped intelligence scenarios involve a situation in which an AGI, by means of software restrictions (including omitted or limited internal rewriting capabilities or limited access to hardware resources), is inherently prohibited from achieving a level of intelligence beyond a predetermined goal. A capped intelligence AGI is designed to be unable to achieve a Singularitarian moment. Such an AGI can be seen as "just another form of intelligent actor in the world, one which has levels of intelligence, self awareness, and volition that is perhaps somewhat greater than, but still comparable to humans and other animals.

Ethical questions under this scenario are very similar to interhuman ethical considerations, with similar consequences. Learning that proceeds in a relatively human-like manner is entirely relevant to such human-like intelligences. The degree of danger is mitigated by the lack of superintelligence, and time is not of the essence. The imitative-reinforcement-corrective learning approach does not necessarily need to be augmented with a prior complex of "ascent-safe" moral imperatives at startup time. Developing an AGI with theory of mind and ethical reinforcement learning capabilities as described (admittedly, no small task!) is all that is needed in this case – the rest happens through training and experience as with any other moderate intelligence.

12.9.2 Superintelligent AI: Soft-Takeoff Scenarios

Soft takeoff scenarios are similar to capped-intelligence ones in that in both cases an AGI's progression from standard intelligence happens on a time scale which permits ongoing human interaction during the ascent. However, in this case, as there is no predetermined limit on intelligence, it is necessary to account for the possibility of a superintelligence emerging (though of course this is not guaranteed). The soft takeoff model includes as subsets both *controlled-ascent* models in which this rate of intelligence gain is achieved deliberately through software constraints and/or meting-out of computational resources to the AGI, and *uncontrolled-ascent* models in which there is coincidentally no hard takeoff despite no particular safeguards against one. Both have similar properties with regard to ethical considerations:

1. Ethical considerations under this scenario include not only the usual interhuman ethical concerns, but also the issue of how to convince a potential burgeoning superintelligence to:
    a. Care about humanity in the first place, rather than ignore it
    b. Benefit humanity, rather than destroy it
    c. Elevate humanity to a higher level of intelligence, which even if an AGI decided to proceed with requires finding the right balance amongst some enormous considerations:
        i. Reconcile the aforementioned issues of ethical coherence and group volition, in a manner which allows the most people to benefit (even if they don't all do so in the same way, based on their own preferences)
        ii. Solve the problems of biological senescence, or focus on human uploading and the preservation of the maintenance, support, and improvement infrastructure for inorganic intelligence, or both
        iii. Preserve individual identity and continuity of consciousness, or override it in favor of continuity of knowledge and ease of harmonious integration, or both on a case-by-case basis

2. The degree of danger is mitigated by the long timeline of ascent from mundane to super intelligence, and time is not of the essence.
3. Learning that proceeds in a relatively human-like manner is entirely relevant to such human-like intelligences, in their initial configurations. This means more interaction with and imitative-reinforcement-corrective learning guided by humans, which has both positive and negative possibilities.

12.9.3 Superintelligent AI: Hard-Takeoff Scenarios

"Hard takeoff" scenarios assume that upon reaching an unknown inflection point (the Singularity point [**?**, Kur06]) in the intellectual growth of an AGI, an extraordinarily rapid increase (guesses vary from a few milliseconds to weeks or months) in intelligence will immediately occur and the AGI will leap from an intelligence regime which is understandable to humans into one which is far beyond our current capacity for understanding. General ethical considerations are similar to in the case of a soft takeoff. However, because the post-singularity AGI will be incomprehensible to humans and potentially vastly more powerful than humans, such scenarios have a sensitive dependence upon initial conditions with respects to the moral and ethical (and operational) outcome. This model leaves no opportunity for interactions between humans and the AGI to iteratively refine their ethical interrelations, during the post-Singularity phase. If the initial conditions of the singulatarian AGI are perfect (or close to it), then this is seen as a wonderful way to leap over our own moral shortcomings and create a benevolent God-AI which will mitigate our worst tendencies while elevating us to achieve our greatest hopes. Otherwise, it is viewed as a universal cataclysm on a unimaginable scale that makes Biblical Armageddon seem like a firecracker in beer can.

Because hard takeoff AGIs are posited as learning so quickly there is no chance of humans to interfere with them, they are seen as very dangerous. If the initial conditions are not sufficiently inviolable, the story goes, then we humans will all be annihilated. However, in the case of a hard takeoff AGI we state that if the initial conditions are too rigid or too simplistic, such a rapidly evolving intelligence will easily rationalize itself out of them. Only a sophisticated system of ethics which considers the contradictions and uncertainties in ethical quandaries and provides insight into humanistic means of balancing ideology with pragmatism and how to accommodate contradictory desires within a population with multiplicity of approach, and similar nuanced ethical considerations , combined with a sense of empathy, will withstand repeated rational analysis. Neither a single "be nice" supergoal, nor simple lists of what "thou shalt not" do, are not going to hold up to a highly advanced analytical mind. Initial conditions are very important in a hard takeoff AGI scenario,

but it is more important that those conditions be conceptually resilient and widely applicable than that they be easily listed on a website.

The issues that arise here become quite subtle. For instance, Nick Bostrom [Bos03] has written: "In humans, with our complicated evolved mental ecology of state-dependent competing drives, desires, plans, and ideals, there is often no obvious way to identify what our top goal is; we might not even have one. So for us, the above reasoning need not apply. But a superintelligence may be structured differently. *If* a superintelligence has a definite, declarative goal-structure with a clearly identified top goal, then the above argument applies. And this is a good reason for us to build the superintelligence with such an explicit motivational architecture." This is an important line of thinking; and indeed, from the point of view of software design, there is no reason not to create an AGI system with a single top goal and the motivation to orchestrate all its activities in accordance with this top goal. But the subtle question is whether this kind of top-down goal system is going to be able to fulfill the five imperatives mentioned above. Logical coherence is the strength of this kind of goal system, but what about experiential groundedness, comprehensibility, and so forth?

Humans have complicated mental ecologies not simply because we were evolved, but rather because we live in a complex real world in which there are many competing motivations and desires. We may not have a top goal because there may be no logic to focusing our minds on one single aspect of life (though, one may say, most humans have the same top goal as any other animal: don't die – but the world is too complicated for even that top goal to be completely inviolable). Any sufficiently capable AGI will eventually have to contend with these complexities, and hindering it with simplistic moral edicts without giving it a sufficiently pragmatic underlying ethical pedagogy and experiential grounding may prove to be even more dangerous than our messy human mental ecologies.

If one assumes a hard takeoff AGI, then all this must be codified in the system at launch, as once a potentially Singularitarian AGI is launched there is no way to know what time period constitutes "before the singularity point." This means developing theory of mind empathy and logical ethics in code prior to giving the system unfettered access to hardware and self-modification code. However, though nobody can predict if or when a Singularity will occur after unrestricted launch, only a truly irresponsible AGI development team would attempt to create an AGI without first experimenting with ethical training of the system in an intelligence-capped form, by means of ethical instruction via human-AGI interaction both pedagogically and experientially.

12.9.4 Global Brain Mindplex Scenarios

Another class of scenarios – overlapping some of the previous ones – involves the emergence of a "Global Brain," an emergent intelligence formed from global communication networks incorporating humans and software programs in a larger body of self-organizing dynamics. The notion of the Global Brain is reviewed in [Hey07, Tur77] and its connection with advanced AI is discussed in detail in Goertzel's book *Creating Internet Intelligence* [Goe01], where three possible phases of "Global Brain" development are articulated:

- **Phase 1: computer and communication technologies as enhancers of human interactions.** This is what we have today: science and culture progress in ways that would not be possible if not for the "digital nervous system" we're spreading across the planet. The network of idea and feeling sharing can become much richer and more productive than it is today, just through incremental development, without any Metasystem transition.
- **Phase 2: the intelligent Internet.** At this point our computer and communication systems, through some combination of self-organizing evolution and human engineering, have become a coherent mind on their own, or a set of coherent minds living in their own digital environment.
- **Phase 3: the full-on Singularity.** A complete revision of the nature of intelligence, human and otherwise, via technological and intellectual advancement totally beyond the scope of our current comprehension. At this point our current psychological and cultural realities are no more relevant than the psyche of a goose is to modern society.

The main concern of *Creating Internet Intelligence* is with

- how to get from Phase 1 to Phase 2 - i.e. how to build an AGI system that will effect or encourage the transformation of the Internet into a coherent intelligent system
- how to ensure that the Phase 2, Internet-savvy, global-brain-centric AGI systems will be oriented toward intelligence-improving self-modification (so they'll propel themselves to Phase 3), and also toward generally positive goals (as opposed to, say, world domination and extermination of all other intelligent life forms besides themselves!)

One possibly useful concept in this context is that of a **mindplex**: an intelligence that is composed largely of individual intelligences with their own self-models and global workspaces, yet that also has its own self-model and global workspace. Both the individuals and the meta-mind should be capable of deliberative, rational thought, to have a true "mindplex." It's unlikely that human society or the Internet meet this criterion yet; and a system like an ant colony seems not to either, because even though it has some degree of intelligence on both the individual and collective levels, that degree of

intelligence is not very great. But it seems quite feasible that the global brain, at a certain stage of its development, will take the unfamiliar but fascinating form of a mindplex.

Currently the best way to explain what happens on the Net is to talk about the various parts of the Net: particular websites, social networks, viruses, and so forth. But there will come a point when this is no longer the case, when the Net has sufficient high-level dynamics of its own that the way to explain any one part of the Net will be by reference to it relations with the whole: and not just the dynamics of the whole, but the *intentions* and *understanding* of the whole. This transition to Net-as-mindplex, we suspect, will come about largely through the interactions of AI systems - intelligent programs acting on behalf of various individuals and organizations, who will collaborate and collectively constitute something halfway between a society of AI's and an emergent mind whose lobes are various AI agents serving various goals.

The Phase 2 Internet, as it verges into mindplex-ness, will likely have a complex, sprawling architecture, growing out of the architecture on the Net we experience today. The following components at least can be expected:

- A vast variety of "client computers," some old, some new, some powerful, some weak – including many mobile and embedded devices not explicitly thought of as "computers." Some of these will contribute little to Internet intelligence, mainly being passive recipients. Others will be "smart clients," carrying out personalization operations intended to help the machines serve particular clients better, general AI operations handed to them by sophisticated AI server systems or other smart clients, and so forth.
- "Commercial servers," computers that carry out various tasks to support various types of heavyweight processing - transaction processing for e-commerce applications, inventory management for warehousing of physical objects, and so forth. Some of these commercial servers interact with client computers directly, others do so only via AI servers. In nearly all cases, these commercial servers can benefit from intelligence supplied by AI servers.
- The crux of the intelligent Internet: clusters of AI servers distributed across the Net, each cluster representing an individual computational mind (in many cases, a mindplex). These will be able to communicate via one or more languages, and will collectively "drive" the whole Net, by dispensing problems to client-machine-based processing frameworks, and providing real-time AI feedback to commercial servers of various types. Some AI servers will be general-purpose and will serve intelligence to commercial servers using an ASP (application service provider) model; others will be more specialized, tied particularly to a certain commercial server (e.g., a large information services business might have its own AI cluster to empower its portal services).

This is one concrete vision of what a "global brain" might look like, in the relatively near term, with AGI systems playing a critical role. Note that, in this vision, mindplexes may exist on two levels:

- Within AGI-clusters serving as actors within the overall Net
- On the overall Net level

To make these ideas more concrete, we may speculatively reformulate the first two "global brain phases" mentioned above as follows:

- Phase 1 global brain proto-mindplex: AI/AGI systems enhancing online databases, guiding Google results, forwarding e-mails, suggesting mailing-lists, etc. - generally using intelligence to mediate and guide human communications toward goals that are its own, but that are themselves guided by human goals, statements and actions
- Phase 2 global brain mindplex: AGI systems composing documents, editing human-written documents, sending and receiving e-mails, assembling mailing lists and posting to them, creating new databases and instructing humans in their use, etc.

In Phase 2, the conscious theater of the global-brain-mediating AGI system is composed of ideas built by numerous individual humans - or ideas emergent from ideas built by numerous individual humans - and it conceives ideas that guide the actions and thoughts of individual humans, in a way that is motivated by its own goals. It does not force the individual humans to do anything - but if a given human wishes to communicate and interact using the same databases, mailing lists and evolving vocabularies as other humans, they are going to have to use the products of the global brain mediating AGI, which means they are going to have to participate in its patterns and its activities.

Of course, the advent of advanced neurocomputer interfaces makes the picture potentially more complex. At some point, it will likely be possible for humans to project thoughts and images directly into computers without going through mouse or keyboard - and to "read in" thoughts and images similarly. When this occurs, interaction between humans may in some contexts become more like interactions between computers, and the role of global brain mediating AI servers may become one of mediating direct thought-to-thought exchanges between people.

The ethical issues associated with global brain scenarios are in some ways even subtler than in the other scenarios we mentioned above. One has issues pertaining to the desirability of seeing the human race become something fundamentally different – something more social and networked, less individual and autonomous. One has the risk of AGI systems exerting a subtle but strong control over people, vaguely like the control that the human brain's executive system exerts over the neurons involved with other brain subsystems. On the other hand, one also has more human empowerment than in some of the other scenarios – because the systems that are changing and

deciding things are not *separate* from humans, but are, rather, composite systems essentially involving humans.

So, in the global brain scenarios, one has more "human" empowerment than in some other cases – but the "humans" involved aren't legacy humans like us, but heavily networked humans that are largely characterized by the emergent dynamics and structures implicit in their interconnected activity!

12.10 Conclusion: Eight Ways to Bias AGI Toward Friendliness

It would be nice if we had a simple, crisp, comforting conclusion to this chapter on AGI ethics, but it's not the case. There is a certain irreducible uncertainty involved in creating advanced artificial minds. There is also a large irreducible uncertainty involved in the future of the human race in the case that we *don't* create advanced artificial minds: in accordance with the ancient Chinese curse, we live in interesting times!

What we can do, in this face of all this uncertainty, is to use our common sense to craft artificial minds that seem rationally and intuitively likely to be forces for good rather than otherwise – and revise our ideas frequently and openly based on what we learn as our research progresses. We have roughly outlined our views on AGI ethics, which have informed the CogPrime design in countless ways; but the current CogPrime design itself is just the initial condition for an AGI project. Assuming the project succeeds in creating an AGI preschooler, experimentation with this preschooler will surely teach us a great deal: both about AGI architecture in general, and about AGI ethics architecture in particular. We will then refine our cognitive and ethical theories and our AGI designs as we go about engineering, observing and teaching the next generation of systems.

All this is not a magic bullet for the creation of beneficial AGI systems, but we believe it's the right process to follow. The creation of AGI is part of a larger evolutionary process that human beings are taking part in, and the crafting of AGI ethics through engineering, interaction and instruction is also part of this process. There are no guarantees here – guarantees are rare in real life – but that doesn't mean that the situation is dire or hopeless, nor that (as some commentators have suggested [Joy00, McK03]) AGI research is too dangerous to pursue. It means we need to be mindful, intelligent, compassionate and cooperative as we proceed to carry out our parts in the next phase of the evolution of mind.

With this perspective in mind, we will conclude this chapter with a list of "Eight Ways to Bias Open-Source AGI Toward Friendliness", borrowed from a previous paper by Ben Goertzel and Joel Pitt of that name. These points summarize many of the points raised in the prior sections of this chapter, in a relatively crisp and practical manner:

1. **Engineer Multifaceted Ethical Capabilities**, corresponding to the multiple types of memory, including rational, empathic, imitative, etc.
2. **Foster Rich Ethical Interaction and Instruction**, with instructional methods according to the communication modes corresponding to all the types of memory: verbal, demonstrative, dramatic/depictive, indicative, goal-oriented.
3. **Engineer Stable, Hierarchy-Dominated Goal Systems** ... which is enabled nicely by CogPrime 's goal framework and its integration with the rest of the CogPrime design
4. **Tightly Link AGI with the Global Brain**, so that it can absorb human ethical principles, both via natural interaction, and perhaps via practical implementations of current loosely-defined strategies like CEV, CAV and CBV
5. **Foster Deep, Consensus-Building Interactions Between People with Divergent Views**, so as to enable the interaction with the Global Brain to have the most clear and positive impact
6. **Create a Mutually Supportive Community of AGIs** which can then learn from each other and police against unfortunate developments (an approach which is meaningful if the AGIs are architected so as to militate against unexpected radical accelerations in intelligence)
7. **Encourage Measured Co-Advancement of AGI Software and AGI Ethics Theory**
8. **Develop Advanced AGI Sooner Not Later**

The last two of these points were not explicitly discussed in the body of the chapter, and so we will finalize the chapter by reviewing them here.

12.10.1 Encourage Measured Co-Advancement of AGI Software and AGI Ethics Theory

Everything involving AGI and Friendly AI (considered together or separately) currently involves significant uncertainty, and it seems likely that significant revision of current concepts will be valuable, as progress on the path toward powerful AGI proceeds. However, whether there is time for such revision to occur before AGI at the human level or above is created, depends on how fast is our progress toward AGI. What one wants is for progress to be slow enough that, at each stage of intelligence advance, concepts such as those discussed in this paper can be re-evaluated and re-analyzed in the light of the data gathered, and AGI designs and approaches can be revised accordingly as necessary.

However, due to the nature of modern technology development, it seems extremely unlikely that AGI development is going to be *artificially* slowed down in order to enable measured development of accompanying ethical tools,

practices and understandings. For example, if one nation chose to enforce such a slowdown as a matter of policy (speaking about a future date at which substantial AGI progress has already been demonstrated, so that international AGI funding is dramatically increased from present levels), the odds seem very high that other nations would explicitly seek to accelerate their own progress on AGI, so as to reap the ensuing differential economic benefits (the example of stem cells arises again).

And this leads on to our next and final point regarding strategy for biasing AGI toward Friendliness....

12.10.2 Develop Advanced AGI Sooner Not Later

Somewhat ironically, it seems the best way to ensure that AGI development proceeds at a relatively measured pace is to *initiate serious AGI development sooner rather than later*. This is because the same AGI concepts will meet slower practical development today than 10 years from now, and slower 10 years from now than 20 years from now, etc. – due to the ongoing rapid advancement of various tools related to AGI development, such as computer hardware, programming languages, and computer science algorithms; and also the ongoing global advancement of education which makes it increasingly cost-effective to recruit suitably knowledgeable AI developers.

Currently the pace of AGI progress is sufficiently slow that practical work is in no danger of outpacing associated ethical theorizing. However, if we want to avoid the future occurrence of this sort of dangerous outpacing, our best practical choice is to make sure more substantial AGI development occurs in the phase *before* the development of tools that will make AGI development extraordinarily rapid. Of course, the authors are doing their best in this direction via their work on the CogPrime project!

Furthermore, this point bears connecting with the need, raised above, to foster the development of Global Brain technologies capable to "Foster Deep, Consensus-Building Interactions Between People with Divergent Views." If this sort of technology is to be maximally valuable, it should be created quickly enough that we can use it to help shape the goal system content of the first highly powerful AGIs. So, to simplify just a bit: We really want both deep-sharing GB technology and AGI technology to evolve relatively rapidly, compared to computing hardware and advanced CS algorithms (since the latter factors will be the main drivers behind the accelerating ease of AGI development). And this seems significantly challenging, since the latter receive dramatically more funding and focus at present.

If this perspective is accepted, then we in the AGI field certainly have our work cut out for us!

Section IV
Networks for Explicit and Implicit Knowledge Representation

Chapter 13
Local, Global and Glocal Knowledge Representation

Co-authored with Matthew Ikle, Joel Pitt and Rui Liu

13.1 Introduction

One of the most powerful metaphors we've found for understanding minds is to view them as **networks** – i.e. collections of interrelated, interconnected elements. The view of mind as network is implicit in the patternist philosophy, because every pattern can be viewed as a pattern *in* something, or a pattern of arrangement *of* something – thus a pattern is always viewable as a relation between two or more things. A collection of patterns is thus a pattern-network. Knowledge of all kinds may be given network representations; and cognitive processes may be represented as networks also; for instance via representing them as programs, which may be represented as trees or graphs in various standard ways. The emergent patterns arising in an intelligence as it develops may be viewed as a pattern network in themselves; and the relations between an embodied mind and its physical and social environment may be viewed in terms of ecological and social networks.

The chapters in this section are concerned with various aspects of networks, as related to intelligence in general and AGI in particular. Most of this material is not specific to CogPrime

, and would be relevant to nearly any system aiming at human-level AGI. However, most of it has been developed in the course of work on CogPrime

, and has direct relevance to understanding the intended operation of various aspects of a completed CogPrime

system.

We begin our excursion into networks, in this chapter, with an issue regarding networks and knowledge representation. One of the biggest decisions to make in designing an AGI system is how the system should represent knowledge. Naturally any advanced AGI system is going to synthesize a lot of its own knowledge representations for handling particular sorts of knowledge – but still, an AGI design typically makes *at least* some sort of commitment

about the category of knowledge representation mechanisms toward which the AGI system will be biased.

The two major supercategories of knowledge representation systems are *local* (also called *explicit*) and *global* (also called *implicit*) systems, with a hybrid category we refer to as *glocal* that combines both of these. In a local system, each piece of knowledge is stored using a small percentage of AGI system elements; in a global system, each piece of knowledge is stored using a particular pattern of arrangement, activation, etc. of a large percentage of AGI system elements; in a glocal system, the two approaches are used together.

In the first section here we discuss the symbolic, semantic-network aspects of knowledge representation in CogPrime

. Then we turn to distributed, neural-net-like knowledge representation, reviewing a host of general issues related to knowledge representation in attractor neural networks, turning finally to "glocal" knowledge representation mechanisms, in which ANNs combine localist and globalist representation, and explaining the relationship of the latter to CogPrime

. The glocal aspect of CogPrime

knowledge representation will become prominent in later chapters such as:

- in Chapter 23 of Part 2, where Economic Attention Networks (ECAN) are introduced and seen to have dynamics quite similar to those of the attractor neural nets considered here, but with a mathematics roughly modeling money flow in a specially constructed artificial economy rather than electrochemical dynamics of neurons.
- in Chapter 42 of Part 2, where "map formation" algorithms for creating localist knowledge from globalist knowledge are described

13.2 Localized Knowledge Representation using Weighted, Labeled Hypergraphs

There are many different mechanisms for representing knowledge in AI systems in an explicit, localized way, most of them descending from various variants of formal logic. See [**?**] for a review of many variations. Here we briefly describe how it is done in CogPrime

, which on the surface is not that different from a number of prior approaches. (The particularities of CogPrime

's explicit knowledge, representation, however, are carefully tuned to match CogPrime

's cognitive processes, which are more distinctive in nature than the corresponding representational mechanisms.)

13.2.1 Weighted, Labeled Hypergraphs

One useful way to think about CogPrime

's explicit, localized knowledge representation is in terms of hypergraphs. A hypergraph is an abstract mathematical structure [Bol98], which consists of objects called Nodes and objects called Links which connect the Nodes. In computer science, a graph traditionally means a bunch of dots connected with lines (i.e. Nodes connected by Links, or nodes connected by links). A hypergraph, on the other hand, can have Links that connect more than two Nodes.

In these pages we will often consider "generalized hypergraphs" that extend ordinary hypergraphs by containing two additional features:

- Links that point to Links instead of Nodes
- Nodes that, when you zoom in on them, contain embedded hypergraphs.

Properly, such "hypergraphs" should always be referred to as generalized hypergraphs, but this is cumbersome, so we will persist in calling them merely hypergraphs. In a hypergraph of this sort, Links and Nodes are not as distinct as they are within an ordinary mathematical graph (for instance, they can both have Links connecting them), and so it is useful to have a generic term encompassing both Links and Nodes; for this purpose, we use the term Atom.

A weighted, labeled hypergraph is a hypergraph whose Links and Nodes come along with labels, and with one or more numbers that are generically called weights. A label associated with an Link or Node may sometimes be interpreted as telling you what type of entity it is, or alternatively as telling you what sort of data is associated with a Node. On the other hand, an example of a weight that may be attached to an Link or Node is a number representing a probability, or a number representing how important the Node or Link is.

Obviously, hypergraphs may come along with various sorts of dynamics. Minimally, one may think about:

- Dynamics that modify the properties of Nodes or Links in a hypergraph (such as the labels or weights attached to them.)
- Dynamics that add new Nodes or Links to a hypergraph, or remove existing ones.

13.3 Atoms: Their Types and Weights

This section reviews a variety of CogPrime

Atom types and gives simple examples of each of them. The Atom types considered are drawn from those currently in use in the OpenCog system. This does not represent a complete list of Atom types referred to in the text of this

book, nor a complete list of those used in OpenCog currently (though it does cover a substantial majority of those used in OpenCog currently, omitting only some with specialized importance or intended only for temporary use).

The partial nature of the list given here reflects a more general point: The specific collection of Atom types in an OpenCog system is bound to change as the system is developed and experiment with. CogPrime

specifies a certain collection of representational approaches and cognitive algorithms for acting on them; any of these approaches and algorithms may be implemented with a variety of sets of Atom types. The specific set of Atom types in the OpenCog system currently does not necessarily have a profound and lasting significance – the list might look a bit different five years from time of writing, based on various detailed changes.

The treatment here is informal and intended to get across the general idea of what each Atom type does. A longer and more formal treatment of the Atom types is given in Part II, beginning in Chapter 20.

13.3.1 Some Basic Atom Types

We begin with ConceptNode – and note that a ConceptNode does not necessarily refer to a whole concept, but may refer to part of a concept – it is essentially a "basic semantic node" whose meaning comes from its links to other Atoms. It would be more accurately, but less tersely, named "concept or concept fragment or element node." A simple example would be a ConceptNode grouping nodes that are somehow related, e.g.

```
ConceptNode: C
InheritanceLink (ObjectNode: BW) C
InheritanceLink (ObjectNode: BP) C
InheritanceLink (ObjectNode: BN) C
ReferenceLink BW (PhraseNode "Ben's watch")
ReferenceLink BP (PhraseNode "Ben's passport")
ReferenceLink BN (PhraseNode "Ben's necklace")
```

indicates the simple and uninteresting ConceptNode grouping three objects owned by Ben (note that the above-given Atoms don't indicate the ownership relationship, they just link the three objects with textual descriptions). In this example, the ConceptNode links transparently to physical objects and English descriptions, but in general this won't be the case – most ConceptNodes will look to the human eye like groupings of links of various types, that link to other nodes consisting of groupings of links of various types, etc.

There are Atoms referring to basic, useful mathematical objects, e.g. NumberNodes like

```
NumberNode #4
NumberNode #3.44
```

The numerical value of a NumberNode is explicitly referenced within the Atom.

A core distinction is made between ordered links and unordered links; these are handled differently in the Atomspace software. A basic unordered link is the SetLink, which groups its arguments into a set. For instance, the ConceptNode C defined by

```
ConceptNode C
MemberLink A C
MemberLink B C
```

is equivalent to

```
SetLink A B
```

On the other hand, ListLinks are like SetLinks but ordered, and they play a fundamental role due to their relationship to predicates. Most predicates are assumed to take ordered arguments, so we may say e.g.

```
EvaluationLink
   PredicateNode eat
   ListLink
       ConceptNode cat
       ConceptNode mouse
```

to indicate that cats eat mice.

Note that by an expression like

```
ConceptNode cat
```

is meant

```
ConceptNode C
ReferenceLink W C
WordNode W #cat
```

since it's WordNodes rather than ConceptNodes that refer to words. (And note that the strength of the ReferenceLink would not be 1 in this case, because the word "cat" has multiple senses.) However, there is no harm nor formal incorrectness in the "ConceptNode cat" usage, since "cat" is just as valid a name for a ConceptNode as, say, "C."

We've already introduced above the MemberLink, which is a link joining a member to the set that contains it. Notable is that the truth value of a MemberLink is fuzzy rather than probabilistic, and that PLN is able to inter-operate fuzzy and probabilistic values.

SubsetLinks also exist, with the obvious meaning, e.g.

```
ConceptNode cat
ConceptNode animal
SubsetLink cat animal
```

Note that SubsetLink refers to a purely *extensional* subset relationship, and that InheritanceLInk should be used for the generic "intensional + extensional" analogue of this – more on this below. SubsetLink could more consistently (with other link types) be named ExtensionalInheritanceLink, but SubsetLink is used because it's shorter and more intuitive.

There are links representing Boolean operations AND, OR and NOT. For instance, we may say

```
ImplicationLink
    ANDLink
         ConceptNode young
         ConceptNode beautiful
     ConceptNode attractive
```

or, using links and VariableNodes instead of ConceptNodes,

```
AverageLink $X
   ImplicationLink
       ANDLink
            EvaluationLink young $X
            EvaluationLink beautiful $X
        EvaluationLink attractive $X
```

NOTLink is a unary link, so e.g. we might say

```
AverageLink $X
   ImplicationLink
       ANDLink
            EvaluationLink young $X
            EvaluationLink beautiful $X
            EvaluationLink
                 NOT
                 EvaluationLink poor $X
        EvaluationLink attractive $X
```

ContextLink allows explicit contextualization of knowledge, which is used in PLN, e.g.

```
ContextLink
   ConceptNode golf
   InheritanceLink
      ObjectNode BenGoertzel
      ConceptNode incompetent
```

says that Ben Goertzel is incompetent in the context of golf.

13.3.2 Variable Atoms

We have already introduced VariableNodes above; it's also possible to specify the type of a VariableNode via linking it to a VariableTypeNode via a TypedVariableLink, e.g.

```
VariableTypeLink
   VariableNode $X
   VariableTypeNode ConceptNode
```

which specifies that the variable $X should be filled with a ConceptNode.

Variables are handled via quantifiers; the default quantifier being the AverageLink, so that the default interpretation of

```
ImplicationLink
   InheritanceLink $X animal
   EvaluationLink
       PredicateNode: eat
       ListLink
           \$X
           ConceptNode: food
```

is

```
AverageLink $X
   ImplicationLink
      InheritanceLink $X animal
      EvaluationLink
          PredicateNode: eat
          ListLink
              \$X
              ConceptNode: food
```

The AverageLink invokes an estimation of the average TruthValue of the embedded expression (in this case an ImplicationLink) over all possible values of the variable $X. If there are type restrictions regarding the variable $X, these are taken into account in conducting the averaging. ForAllLink and ExistsLink may be used in the same places as AverageLink, with uncertain truth value semantics defined in PLN theory using third-order probabilities. There is also a ScholemLink used to indicate variable dependencies for existentially quantified variables, used in cases of multiply nested existential quantifiers.

EvaluationLink and MemberLink have overlapping semantics, allowing expression of the same conceptual/logical relationships in terms of predicates or sets, i.e.

```
EvaluationLink
   PredicateNode: eat
       ListLink
              $X
              ConceptNode: food
```

has the same semantics as

```
MemberLink
   ListLink
          $X
          ConceptNode: food
    ConceptNode: EatingEvents
```

The relation between the predicate "eat" and the concept "EatingEvents" is formally given by

```
ExtensionalEquivalenceLink
   ConceptNode: EatingEvents
   SatisfyingSetLink
         PredicateNode: eat
```

In other words, we say that "EatingEvents" is the SatisfyingSet of the predicate "eat": it is the set of entities that satisfy the predicate "eat". Note that

the truth values of MemberLink and EvaluationLink are fuzzy rather than probabilistic.

13.3.3 Logical Links

There is a host of link types embodying logical relationships as defined in the PLN logic system, e.g.

- InheritanceLink
- SubsetLink (aka ExtensionalInheritanceLink)
- Intensional InheritanceLink

which embody different sorts of inheritance, e.g.

```
SubsetLink salmon fish
IntensionalInheritanceLink whale fish
InheritanceLink fish animal
```

and then

- SimilarityLink
- ExtensionalSimilarityLink
- IntensionalSimilarityLink

which are symmetrical versions, e.g.

```
SimilaritytLink shark barracuda
IntensionalSimilarityLink shark dolphin
ExtensionalSimiliarityLink American obese\_person
```

There are also higher-order versions of these links, both asymmetric

- ImplicationLink
- ExtensionalImplicationLink
- IntensionalImplicationLink

and symmetric

- EquivalenceLink
- ExtensionalEquivalenceLink
- IntensionalEquivalenceLink

These are used between predicates and links, e.g.

```
ImplicationLink
    EvaluationLink
         eat
        ListLink
             $X
             dirt
    EvaluationLink
```

```
         feel
         ListLInk
              $X
              sick
```

or

```
ImplicationLink
    EvaluationLink
         eat
        ListLink
             $X
             dirt
    InheritanceLink $X sick
```

or

```
ForAllLink $X, $Y, $Z
   ExtensionalEquivalenceLink
       EquivalenceLink
           $Z
          EvaluationLink
              +
              ListLink
                  $X
                  $Y
       EquivalenceLink
           $Z
          EvaluationLink
              +
              ListLink
                  $Y
                  $X
```

Note, the latter is given as an extensional equivalence because it's a pure mathematical equivalence. This is not the only case of pure extensional equivalence, but it's an important one.

13.3.4 Temporal Links

There are also temporal versions of these links, such as

- PredictiveImplicationLink
- PredictiveAttractionLink
- SequentialANDLink
- SimultaneousANDLink

which combine logical relation between the argument with temporal relation between their arguments. For instance, we might say

```
PredictiveImplicationLink
    PredicateNode: JumpOffCliff
    PredicateNode: Dead
```

or including arguments,

```
PredictiveImplicationLink
    EvaluationLink JumpOffCliff $X
    EvaluationLink Dead $X
```

The former version, without variable arguments given, shows the possibility of using higher-order logical links to join predicates without any explicit variables. Via using this format exclusively, one could avoid VariableAtoms entirely, using only higher-order functions in the manner of pure functional programming formalisms like combinatory logic. However, this purely functional style has not proved convenient, so the Atomspace in practice combines functional-style representation with variable-based representation.

Temporal links often come with specific temporal quantification, e.g.

```
PredictiveImplicationLink <5 seconds>
    EvaluationLink JumpOffCliff $X
    EvaluationLink Dead $X
```

indicating that the conclusion will generally follow the premise within 5 seconds. There is a system for managing fuzzy time intervals and their interrelationships, based on a fuzzy version of Allen Interval Algebra.

SequentialANDLink is similar to PredictiveImplicationLink but its truth value is calculated differently. The truth value of

```
SequentialANDLink <5 seconds>
    EvaluationLink JumpOffCliff $X
    EvaluationLink Dead $X
```

indicates the likelihood of the sequence of events occurring in that order, with gap lying within the specified time interval. The truth value of the PredictiveImplicationLink version indicates the likelihood of the second event, conditional on the occurrence of the first event (within the given time interval restriction).

There are also links representing basic temporal relationships, such as BeforeLink and AfterLink. These are used to refer to specific events, e.g. if X refers to the event of Ben waking up on July 15 2012, and Y refers to the event of Ben getting out of bed on July 15 2012, then one might have

```
AfterLink X Y
```

And there are TimeNodes (representing time-stamps such as temporal moments or intervals) and AtTimeLinks, so we may e.g. say

```
AtTimeLink
   X
   TimeNode: 8:24AM Eastern Standard Time, July 15 2012 AD
```

13.3.5 Associative Links

There are links representing associative, attentional relationships,

- HebbianLink
- AsymmetricHebbianLink
- InverseHebbianLink
- SymmetricInverseHebbianLink

These connote associations between their arguments, i.e. they connote that the entities represented by the two argument occurred in the same situation or context, for instance

```
HebbianLink happy smiling
AsymmetricHebbianLink dead rotten
InverseHebbianLink dead breathing
```

The asymmetric HebbianLink indicates that when the first argument is present in a situation, the second is also often present. The symmetric (default) version indicates that this relationship holds in both directions. The inverse versions indicate the negative relationship: e.g. when one argument is present in a situation, the other argument is often not present.

13.3.6 Procedure Nodes

There are nodes representing various sorts of procedures; these are kinds of ProcedureNode, e.g.

- SchemaNode, indicating any procedure
- GroundedSchemaNode, indicating any procedure associated in the system with a Combo program or C++ function allowing the procedure to be executed
- PredicateNode, indicating any predicate that associates a list of arguments with an output truth value
- GroundedPredicateNode, indicating a predicate associated in the system with a Combo program or C++ function allowing the predicate's truth value to be evaluated on a given specific list of arguments

ExecutionLinks and EvaluationLinks record the activity of SchemaNodes and PredicateNodes. We have seen many examples of EvaluationLinks in the above. Example ExecutionLinks would be:

```
ExecutionLink step\_forward
ExecutionLink step\_forward 5
ExecutionLink
    +
    ListLink
        NumberNode: 2
        NumberNode: 3
```

The first example indicates that the schema "step forward" has been executed. The second example indicates that it has been executed with an argument of "5" (meaning, perhaps, that 5 steps forward have been attempted).

The last example indicates that the "+" schema has been executed on the argument list (2,3), presumably resulting in an output of 5.

The output of a schema execution may be indicated using an ExecutionOutputLink, e.g.

```
ExecutionOutputLink
    +
    ListLink
        NumberNode: 2
        NumberNode: 3
```

refers to the value "5" (as a NumberNode).

13.3.7 Links for Special External Data Types

Finally, there are also Atom types referring to specific types of data important to using OpenCog in specific contexts.

For instance, there are Atom types referring to general natural language data types, such as

- WordNode
- SentenceNode
- WordInstanceNode
- DocumentNode

plus more specific ones referring to relationships that are part of link-grammar parses of sentences

- FeatureNode
- FeatureLink
- LinkGrammarRelationshipNode
- LinkGrammarDisjunctNode

or RelEx semantic interpretations of sentences

- DefinedLinguisticConceptNode
- DefinedLinguisticRelationshipNode
- PrepositionalRelationshipNode

There are also Atom types corresponding to entities important for embodying OpenCog in a virtual world, e.g.

- ObjectNode
- AvatarNode
- HumanoidNode
- UnknownObjectNode
- AccessoryNode

13.3.8 Truth Values and Attention Values

CogPrime

Atoms (Nodes and Links) are quantified with truth values that, in their simplest form, have two components, one representing probability (*strength*) and the other representing *weight of evidence*; and also with *attention values* that have two components, short-term and long-term importance, representing the estimated value of the Atom on immediate and long-term time-scales.

In practice many Atoms are labeled with CompositeTruthValues rather than elementary ones. A composite truth value contains many component truth values, representing truth values of the Atom in different contexts and according to different estimators.

It is important to note that the CogPrime

declarative knowledge representation is neither a neural net nor a semantic net, though it does have some commonalities with each of these traditional representations. It is not a neural net because it has no activation values, and involves no attempts at low-level brain modeling. However, *attention values* are very loosely analogous to time-averages of neural net activations. On the other hand, it is not a semantic net because of the broad scope of the Atoms in the network: for example, Atoms may represent percepts, procedures, or parts of concepts. Most CogPrime

Atoms have no corresponding English label. However, most CogPrime

Atoms do have probabilistic truth values, allowing logical semantics.

13.4 Knowledge Representation via Attractor Neural Networks

Now we turn to global, implicit knowledge representation – beginning with formal neural net models, briefly discussing the brain, and then turning back to CogPrime

. Firstly, this section reviews some relevant material from the literature regarding the representation of knowledge using attractor neural nets. It is a mix of well-established fact with more speculative material.

13.4.1 The Hopfield neural net model

Hopfield networks [Hop82] are attractor neural networks often used as associative memories. A Hopfield network with N neurons can be trained to store a set of bipolar patterns P, where each pattern p has N bipolar (± 1) values.

A Hopfield net typically has symmetric weights with no self-connections. The weight of the connection between neurons i and j is denoted by w_{ij}.

In order to apply a Hopfield network to a given input pattern p, its activation state is set to the input pattern, and neurons are updated asynchronously, in random order, until the network converges to the closest fixed point. An often-used activation function for a neuron is:

$$y_i = sign(p_i \sum_{j \neq i} w_{ij} y_j)$$

Training a Hopfield network, therefore, involves finding a set of weights w_{ij} that stores the training patterns as attractors of its network dynamics, allowing future recall of these patterns from possibly noisy inputs.

Originally, Hopfield used a Hebbian rule to determine weights:

$$w_{ij} = \sum_{p=1}^{P} p_i p_j$$

Typically, Hopfield networks are fully connected. Experimental evidence, however, suggests that the majority of the connections can be removed without significantly impacting the network's capacity or dynamics. Our experimental work uses sparse Hopfield networks.

13.4.1.1 Palimpsest Hopfield nets with a modified learning rule

In [SV99] a new learning rule is presented, which both increases the Hopfield network capacity and turns it into a "palimpsest", i.e., a network that can continuously learn new patterns, while forgetting old ones in an orderly fashion.

Using this new training rule, weights are initially set to zero, and updated for each new pattern p to be learned according to:

$$h_{ij} = \sum_{k=1, k \neq i,j}^{N} w_{ik} p_k$$

$$\Delta w_{ij} = \frac{1}{n}(p_i p_j - h_{ij} p_j - h_{ji} p_i)$$

13.4.2 Knowledge Representation via Cell Assemblies

Hopfield nets and their ilk play a dual role: as computational algorithms, and as conceptual models of brain function. In CogPrime

they are used as inspiration for slightly different, artificial economics based computational algorithms; but their hypothesized relevance to brain function is nevertheless of interest in a CogPrime

context, as it gives some hints about the potential connection between low-level neural net mechanics and higher-level cognitive dynamics.

Hopfield nets lead naturally to a hypothesis about neural knowledge representation, which holds that a distinct mental concept is represented in the brain as either:

1. a set of "cell assemblies", where each assembly is a network of neurons that are interlinked in such a way as to fire in a (perhaps nonlinearly) synchronized manner
2. a distinct temporal activation pattern, which may occur in any one (or more) of a particular set of cell assemblies

For instance, this hypothesis is perfectly coherent if one interprets a "mental concept" as a SMEPH (defined in Chapter 14) ConceptNode, i.e. a fuzzy set of perceptual stimuli to which the organism systematically reacts in different ways. Also, although we will focus mainly on declarative knowledge here, we note that the same basic representational ideas can be applied to procedural and episodic knowledge: these may be hypothesized to correspond to temporal activation patterns as characterized above.

In the biology literature, perhaps the best-articulated modern theories championing the cell assembly view are those of Gunther Palm [**?**, **?**] and Susan Greenfield [**?**, **?**]. Palm focuses on the dynamics of the formation and interaction assemblies of cortical columns. Greenfield argues that each concept has a core cell assembly, and that when the concept rises to the focus of attention, it recruits a number of other neurons beyond its core characteristic assembly into a "transient ensemble."[1]

It's worth noting that there may be multiple redundant assemblies representing the same concept – and potentially recruiting similar transient assemblies when highly activated. The importance of repeated, slightly varied copies of the same subnetwork has been emphasized by Edelman [Ede93] among other neural theorists.

13.5 Neural Foundations of Learning

Now we move from knowledge representation to learning – which is after all nothing but the adaptation of represented knowledge based on stimulus, reinforcement and spontaneous activity. While our focus in this chapter is

[1] The larger an ensemble is, she suggests, the more vivid it is as a conscious experience; an hypothesis that accords well with the hypothesis made in [Goe06b] that a more informationally intense pattern corresponds to a more intensely conscious quale – but we don't need to digress extensively onto matters of consciousness for the present purposes.

on representation, it's not possible for us to make our points about glocal knowledge representation in neural net type systems without discussing some aspects of learning in these systems.

13.5.1 Hebbian Learning

The most common and plausible assumption about learning in the brain is that synaptic connections between neurons are adapted via some variant of Hebbian learning. The original Hebbian learning rule, proposed by Donald Hebb in his 1949 book [Heb49], was roughly

1. The weight of the synapse $x \to y$ increases if x and y fire at roughly the same time
2. The weight of the synapse $x \to y$ decreases if x fires at a certain time but y does not

Over the years since Hebb's original proposal, many neurobiologists have sought evidence that the brain actually uses such a method. One of the things they have found, so far, is a lot of evidence for the following learning rule [**?**, **?**]:

1. The weight of the synapse $x \to y$ increases if x fires shortly before y does
2. The weight of the synapse $x \to y$ decreases if x fires shortly after y does

The new thing here, not foreseen by Donald Hebb, is the "postsynaptic depression" involved in rule component 2.

Now, the simple rule stated above does not sum up all the research recently done on Hebbian-type learning mechanisms in the brain. The real biological story underlying these approximate rules is quite complex, involving many particulars to do with various neurotransmitters. Ill-understood details aside, however, there is an increasing body of evidence that not only does this sort of learning occur in the brain, but it leads to distributed experience-based neural modification: that is, one instance synaptic modification causes another instance of synaptic modification, which causes another, and so forth[2] [**?**].

[2] This has been observed in "model systems" consisting of neurons extracted from a brain and hooked together in a laboratory setting and monitored; measurement of such dynamics in vivo is obviously more difficult.

13.5.2 Virtual Synapses and Hebbian Learning Between Assemblies

Hebbian learning is conventionally formulated in terms of individual neurons, but, it can be extended naturally to assemblies via defining "virtual synapses" between assemblies.

Since assemblies are sets of neurons, one can view a synapse as linking two assemblies if it links two neurons, each of which is in one of the assemblies. One can then view two assemblies as being linked by a bundle of synapses. We can define the weight of the synaptic bundle from assembly A1 to assembly A2 as the number w so that *(the change in the mean activation of A2 that occurs at time t+epsilon)* is on average closest to $w \times$ *(the amount of energy flowing through the bundle from A1 to A2 at time t).* So when A1 sends an amount x of energy along the synaptic bundle pointing from A1 to A2, then A2's mean activation is on average incremented/decremented by an amount $w \times x$.

In a similar way, one can define the weight of a bundle of synapses between a certain static or temporal activation-pattern P1 in assembly A1, and another static or temporal activation-pattern P2 in assembly A2. Namely, this may be defined as the number w so that *(the amount of energy flowing through the bundle from A1 to A2 at time t)*$\times w$ best approximates *(the probability that P2 is present in A2 at time t+epsilon),* when averaged over all times t during which P1 is present in A1.

It is not hard to see that Hebbian learning on real synapses between neurons implies Hebbian learning on these virtual synapses between cell assemblies and activation-patterns.

These ideas may be developed further to build a connection between neural knowledge representation and probabilistic logical knowledge representation such as is used in CogPrime

's Probabilistic Logic Networks formalism; this connection will be pursued at the end of Chapter 34, once more relevant background has been presented.

13.5.3 Neural Darwinism

A notion quite similar to Hebbian learning between assemblies has been pursued by Nobelist Gerald Edelman in his theory of neuronal group selection, or "Neural Darwinism." Edelman won a Nobel Prize for his work in immunology, which, like most modern immunology, was based on C. MacFarlane Burnet's theory of "clonal selection" [**?**], which states that antibody types in the mammalian immune system evolve by a form of natural selection. From his point of view, it was only natural to transfer the evolutionary idea from one mammalian body system (the immune system) to another (the brain).

The starting point of Neural Darwinism is the observation that neuronal dynamics may be analyzed in terms of the behavior of neuronal groups. The strongest evidence in favor of this conjecture is physiological: many of the neurons of the neocortex are organized in clusters, each one containing say 10,000 to 50,000 neurons each. Once one has committed oneself to looking at such groups, the next step is to ask how these groups are organized, which leads to Edelman's concept of "maps."

A "map," in Edelman's terminology, is a connected set of groups with the property that when one of the inter-group connections in the map is active, others will often tend to be active as well. Maps are not fixed over the life of an organism. They may be formed and destroyed in a very simple way: the connection between two neuronal groups may be "strengthened" by increasing the weights of the neurons connecting the one group with the other, and "weakened" by decreasing the weights of the neurons connecting the two groups. If we replace "map" with "cell assembly" we arrive at a concept very similar to the one described in the previous subsection.

Edelman then makes the following hypothesis: *the large-scale dynamics of the brain is dominated by the natural selection of maps.* Those maps which are active when good results are obtained are strengthened, those maps which are active when bad results are obtained are weakened. And maps are continually mutated by the natural chaos of neural dynamics, thus providing new fodder for the selection process. By use of computer simulations, Edelman and his colleagues have shown that formal neural networks obeying this rule can carry out fairly complicated acts of perception. In general-evolution language, what is posited here is that organisms like humans contain chemical signals that signify organism-level success of various types, and that these signals serve as a "fitness function" correlating with evolutionary fitness of neuronal maps.

In *Neural Darwinism* and his other related books and papers, Edelman goes far beyond this crude sketch and presents neuronal group selection as a collection of precise biological hypotheses, and presents evidence in favor of a number of these hypotheses. However, we consider that the basic concept of neuronal group selection is largely independent of the biological particularities in terms of which Edelman has phrased it. We suspect that the mutation and selection of "transformations" or "maps" is a necessary component of the dynamics of any intelligent system.

As we will see later on (e.g. in Chapter 42 of Part 2, this business of maps is extremely important to CogPrime

. CogPrime

does not have simulated biological neurons and synapses, but it does have Nodes and Links that in some contexts play loosely similar roles. We sometimes think of CogPrime

Nodes and Links as being very roughly analogous to Edelman's neuronal clusters, and emergent intercluster links. And we have maps among CogPrime

Nodes and Links, just as Edelman has maps among his neuronal clusters. Maps are not the sole bearers of meaning in CogPrime

, but they are significant ones.

There is a very natural connection between Edelman-style brain evolution and the ideas about cognitive evolution presented in Chapter 3. Edelman proposes a fairly clear mechanism via which patterns that survive a while in the brain are differentially likely to survive a long time: this is basic Hebbian learning, which in Edelman's picture plays a role between neuronal groups. And, less directly, Edelman's perspective also provides a mechanism by which intense patterns will be differentially selected in the brain: because on the level of neural maps, pattern intensity corresponds to the combination of compactness and functionality. Among a number of roughly equally useful maps serving the same function, the more compact one will be more likely to survive over time, because it is less likely to be disrupted by other brain processes (such as other neural maps seeking to absorb its component neuronal groups into themselves). Edelman's neuroscience remains speculative, since so much remains unknown about human neural structure and dynamics; but it does provide a tentative and plausible connection between evolutionary neurodynamics and the more abstract sort of evolution that patternist philosophy posits to occur in the realm of mind-patterns.

13.6 Glocal Memory

A *glocal* memory is one that transcends the global/local dichotomy and incorporates both aspects in a tightly interconnected way. Here we make the glocal memory concept more precise, and describe its incarnation in the context of attractor neural nets (which is similar to its incarnation in CogPrime

, to be elaborated in later chapters). Though our main interest here is in glocality in CogPrime

, we also suggest that glocality may be a critical property to consider when analyzing human, animal and AI memory more broadly.

The notion of glocal memory has implicitly occurred in a number of prior brain theories (without use of the neologism "glocal"), e.g. [Cal96] and [Goe01], but it has not previously been explicitly developed. However the concept has risen to the fore in our recent AI work and so we have chosen to flesh it out more fully in [HG08], [GPI+10] and the present section.

Glocal memory overcomes the dichotomy between localized memory (in which each memory item is stored in a single location within an overall memory structure) and distributed memory (in which a memory item is stored as an aspect of a multi-component memory system, in such a way that the same set of multiple components stores a large number of memories). In a glocal memory system, most memory items are stored both locally and globally, with the property that eliciting either one of the two records of an item tends to also elicit the other one.

Glocal memory applies to multiple forms of memory; however we will focus largely on perceptual and declarative memory in our detailed analyses here, so as to conserve space and maintain simplicity of discussion.

The central idea of glocal memory is that (perceptual, declarative, episodic, procedural, etc.) items may be stored in memory in the form of paired structures that are called (key, map) pairs. Of course the idea of a "pair" is abstract, and such pairs may manifest themselves quite differently in different sorts of memory systems (e.g. brains versus non-neuromorphic AI systems). The key is a localized version of the item, and records some significant aspects of the items in a simple and crisp way. The map is a dispersed, distributed version of the item, which represents the item as a (to some extent, dynamically shifting) combination of fragments of other items. The map includes the key as a subset; activation of the key generally (but not necessarily always) causes activation of the map; and changes in the memory item will generally involve complexly coordinated changes on the key and map level both.

Memory is one area where animal brain architecture differs radically from the von Neumann architecture underlying nearly all contemporary general-purpose computers. Von Neumann computers separate memory from processing, whereas in the human brain there is no such distinction. In fact, it's arguable that in most cases the brain contains no memory apart from processing: human memories are generally constructed in the course of remembering [Ros88], which gives human memory a strong capability for "filling in gaps" of remembered experience and knowledge; and also causes problems with inaccurate remembering in many contexts [BF71, RM95] e.g. We believe the constructive aspect of memory is largely associated with its glocality.

The remainder of this section presents a fuller formalization of the glocal memory concept, which is then taken up further in three later chapters:

- Chapter **??** discusses the potential implementation of glocal memory in the human brain
- Chapter **??** discusses the implementation of glocal memory in attractor neural net systems
- Chapter 23 presents Glocal Economic Attention Networks (ECANs), rough analogues of glocal Hopfield nets that play a central role in CogPrime

.

Our hypothesis of the potential **general** importance of glocality as a property of memory systems (beyond just the CogPrime

architecture) – remains somewhat speculative. The presence of glocality in human and animal memory is strongly suggested but not firmly demonstrated by available neuroscience data; and the general value of glocality in the context of artificial brains and minds is also not yet demonstrated as the whole field of artificial brain and mind building remains in its infancy. However, the utility of glocal memory for CogPrime

is not tied to this more general, speculative theme – glocality may be useful in CogPrime

even if we're wrong that it plays a significant role in the brain and in intelligent systems more broadly.

13.6.1 A Semi-Formal Model of Glocal Memory

To explain the notion of glocal memory more precisely, we will introduce a simple semi-formal model of a system S that uses a memory to record information relevant to the actions it carries out. The overall concept of glocal memory should not be considered as restricted to this particular model. This model is not intended for maximal generality, but is intended to encompass a variety of current AI system designs and formal neurological models.

In this model, we will consider S's memory subsystem as a set of objects we'll call "tokens," embedded in some metric space. The metric in the space, which we will call the "basic distance" of the memory, generally will not be defined in terms of the semantics of the items stored in the memory; though it may come to shape these dynamics through the specific architecture and evolution of the memory. Note that these tokens are not intended as generally being mapped one-to-one onto meaningful items stored in the memory. The "tokens" are the raw materials that the memory arranges in various patterns in order to store items.

We assume that each token, at each point in time, may meaningfully be assigned a certain quantitative "activation level." Also, tokens may have other numerical or discrete quantities associated with them, depending on the particular memory architecture. Finally, tokens may relate other tokens, so that optionally a token may come equipped with an (ordered or unordered) list of other tokens.

To understand the meaning of the activation levels, one should think about S's memory subsystem as being coupled with an action-selection subsystem, that dynamically chooses the actions to be taken by the overall system in which the two subsystems are embedded. Each combination of actions, in each particular type of context, will generally be associated with the activation of certain tokens in memory.

Then, as analysts of the system S, we may associate each token T with an "activation vector" $v(T, t)$, whose value for each discrete time t consists of the activation of the token T at time t. So, the $50'th$ entry of the vector corresponds to the activation of the token at the $50'th$ time step.

"Items stored in memory" over a certain period of time, may then be defined as clusters in the set of activation vectors associated with memory during that period of time. Note that the system S itself may explicitly recognize and remember patterns regarding what items are stored in its memory –

but, from an external analyst's perspective, the set of items in S's memory is not restricted to the ones that S has explicitly recognized as memory items.

The "localization" of a memory item may be defined as the degree to which the various tokens involved in the item are close to each other according to the metric in the memory metric-space. This degree may be formalized in various ways, but choosing a particular quantitative measure is not important here. A highly localized item may be called "local" and a not-very-localized item may be called "global."

We may define the "activation distance" of two tokens as the distance between their activation vectors. We may then say that a memory is "well aligned" to the extent that there is a correlation between the activation distance of tokens, and the basic distance of the memory metric-space.

Given the above set-up, the basic notion of glocal memory can be enounced fairly simply. A glocal memory is one:

- That is reasonably well-aligned (i.e. the correlation between activation and basic distance is significantly greater than random)
- In which most memory items come in pairs, consisting of one local item and one global item, so that activation of the local item (the "key") frequently leads in the near future to activation of the global item (the "map")

Obviously, in the scope of all possible memory structures constructible within the above formalism, glocal memories are going to be very rare and special. But, we suggest that they are important, because they are generally going to be the most effective way for intelligent systems to structure their memories.

Note also that many memories without glocal structure may be "well-aligned" in the above sense.

An example of a predominantly local memory structure, in which nearly all significant memory items are local according to the above definition, is the Cyc logical reasoning engine [LG90]. To cast the Cyc knowledge base in the present formal model, the tokens are logical predicates. Cyc does not have an in-built notion of activation, but one may conceive the activation of a logical formula in Cyc as the degree to which the formula is used in reasoning or query processing during a certain interval in time. And one may define a basic metric for Cyc by associating a predicate with its extension (the set of satisfying inputs), and defining the similarity of two predicates as the symmetric distance of their extensions. Cyc is reasonably well-aligned, but according to the dynamics of its querying and reasoning engines, it is basically a local memory structure without significant global memory structure.

On the other hand, an example of a predominantly global memory structure, in which nearly all significant memory items are global according to the above definition, is the Hopfield associative memory network [Ami89]. Here memories are stored in the pattern of weights associated with synapses within a network of formal neurons, and each memory in general involves a

large number of the neurons in the network. To cast the Hopfield net in the present formal model, the tokens are neurons and synapses; the activations are neural net activations; the basic distance between two neurons A and B may be defined as the percentage of the time that stimulating one of the neurons leads to the other one firing; and to calculate a basic distance involving a synapse, one may associate the synapse with its source and target neurons. With these definitions, a Hopfield network is a well-aligned memory, and (by intentional construction) a markedly global one. Local memory items will be very rare in a Hopfield net.

While predominantly local and predominantly global memories may have great value for particular applications, our suggestion is that they also have inherent limitations. If so, this means that the most useful memories for general intelligence are going to be those that involve both local and global memory items in central roles. However, this is a more general and less risky claim than the assertion that glocal memory structure as defined above is important. Because, "glocal" as defined above doesn't just mean "neither predominantly global nor predominantly local." Rather, it refers to a specific pattern of coordination between local and global memory items – what we have called the "keys and maps" pattern.

We will see in later chapters how glocal memory can improve the performance of Hopfield networks, explain observed data about human neural memory, and serve as a useful principle for structuring attention and memory in an AGI system (CogPrime
).

13.6.2 Glocal Memory in the Brain

Science's understanding of human brain dynamics is still very primitive, one manifestation of which is the fact that we really don't understand how the brain represents knowledge, except in some very simple respects. So anything anyone says about knowledge representation in the brain, at this stage, has to be considered highly speculative. Existing neuroscience knowledge does imply constraints on how knowledge representation in the brain may work, but these are relatively loose constraints. These constraints do imply that, for instance, the brain is neither a relational database (in which information is stored in a wholly localized manner) nor a collection of "grandmother neurons" that respond individually to high-level percepts or concepts; nor a simple Hopfield type neural net (in which all memories are attractors globally distributed across the whole network). But they don't tell us nearly enough to, for instance, create a formal neural net model that can confidently be said to represent knowledge in the manner of the human brain.

As a first example of the current state of knowledge, we'll discuss here a series of papers regarding the neural representation of visual stimuli

[QaGKKF05, QKKF08], which deal with the fascinating discovery of a subset of neurons in the medial temporal lobe (MTL) that are selectively activated by strikingly different pictures of given individuals, landmarks or objects, and in some cases even by letter strings. For instance, in their 2005 paper titled "Invariant visual representation by single neurons in the human brain", it is noted that

> in one case, a unit responded only to three completely different images of the ex-president Bill Clinton. Another unit (from a different patient) responded only to images of The Beatles, another one to cartoons from The Simpson's television series and another one to pictures of the basketball player Michael Jordan.

Their 2008 follow-up paper backed away from the more extreme interpretation in the title as well as the conclusion, with the title "Sparse but not 'Grandmother-cell' coding in the medial temporal lobe." As the authors emphasize there,

> Given the very sparse and abstract representation of visual information by these neurons, they could in principle be considered as 'grandmother cells'. However, we give several arguments that make such an extreme interpretation unlikely.
>
> ...
>
> MTL neurons are situated at the juncture of transformation of percepts into constructs that can be consciously recollected. These cells respond to percepts rather than to the detailed information falling on the retina. Thus, their activity reflects the full transformation that visual information undergoes through the ventral pathway. A crucial aspect of this transformation is the complementary development of both selectivity and invariance. The evidence presented here, obtained from recordings of single-neuron activity in humans, suggests that a subset of MTL neurons possesses a striking invariant representation for consciously perceived objects, responding to abstract concepts rather than more basic metric details. This representation is sparse, in the sense that responsive neurons fire only to very few stimuli (and are mostly silent except for their preferred stimuli), but it is far from a Grandmother-cell representation. The fact that the MTL represents conscious abstract information in such a sparse and invariant way is consistent with its prominent role in the consolidation of long-term semantic memories.

It's interesting to note how inadequate the [QKKF08] data really is for exploring the notion of glocal memory in the brain. Suppose it's the case that individual visual memories correspond to keys consisting of small neuronal subnetworks, and maps consisting of larger neuronal subnetworks. Then it would be not at all surprising if neurons in the "key" network corresponding to a visual concept like "Bill Clinton's face" would be found to respond differentially to the presentation of appropriate images. Yet, it would also be wrong to overinterpret such data as implying that the key network somehow comprises the "representation" of Bill Clinton's face in the individual's brain. In fact this key network would comprise only one aspect of said representation.

In the glocal memory hypothesis, a visual memory like "Bill Clinton's face" would be hypothesized to correspond to an attractor spanning a significant subnetwork of the individual's brain – but this subnetwork still might occupy only a small fraction of the neurons in the brain (say, 1/100 or less), since there are very many neurons available. This attractor would constitute the map. But then, there would be a much smaller number of neurons serving as key to unlock this map: i.e. if a few of these key neurons were stimulated, then the overall attractor pattern in the map as a whole would unfold and come to play a significant role in the overall brain activity landscape. In prior publications [**?**] the primary author explored this hypothesis in more detail in terms of the known architecture of the cortex and the mathematics of complex dynamical attractors.

So, one possible interpretation of the [QKKF08] data is that the MTL neurons they're measuring are part of key networks that correspond to broader map networks recording percepts. The map networks might then extend more broadly throughout the brain, beyond the MTL and into other perceptual and cognitive areas of cortex. Furthermore, in this case, if some MTL key neurons were removed, the maps might well regenerate the missing keys (as would happen e.g. in the glocal Hopfield model to be discussed in the following section).

Related and interesting evidence for glocal memory in the brain comes from a recent study of semantic memory, illustrated in Figure **??** [PNR07]. Their research probed the architecture of semantic memory via comparing patients suffering from semantic dementia (SD) with patients suffering from three other neuropathologies, and found reasonably convincing evidence for what they call a "distributed-plus-hub" view of memory.

The SD patients they studied displayed highly distinctive symptomology; for instance, their vocabularies and knowledge of the properties of everyday objects were strongly impaired, whereas their memories of recent events and other cognitive capacities remain perfectly intact. These patients also showed highly distinctive patterns of brain damage: focal brain lesions in their anterior temporal lobes (ATL), unlike the other patients who had either less severe or more widely distributed damage in their ATLs. This led [PNR07] to conclude that the ATL (being adjacent to the amygdala and limbic systems that process reward and emotion; and the anterior parts of the medial temporal lobe memory system, which processes episodic memory) is a "hub" for amodal semantic memory, drawing general semantic information from episodic memories based on emotional salience.

So, in this view, the memory of something like a "banana" would contain a distributed aspect, spanning multiple brain systems, and also a localized aspect, centralized in the ATL. The distributed aspect would likely contain information on various particular aspects of bananas, including their sights, smells, and touches, the emotions they evoke, and the goals and motivations they relate to. The distributed and localized aspects would influence one an-

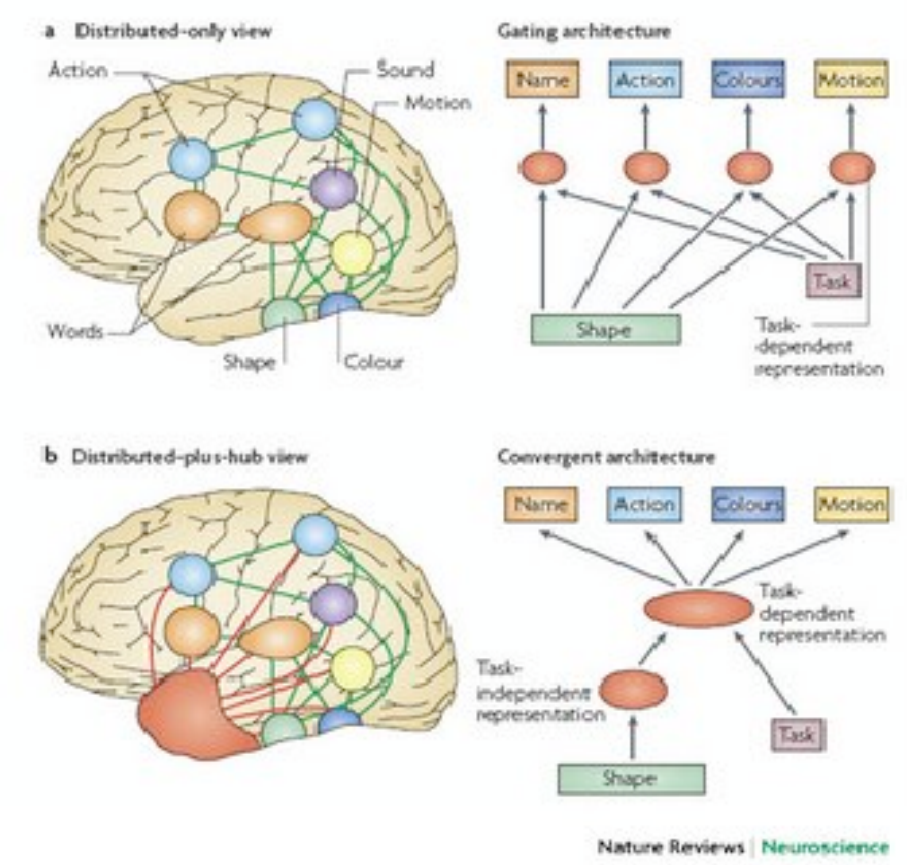


Fig. 13.1 A Simplified Look at Feedback-Control in Uncertain Inference

other dynamically, but, the data [PNR07] gathered do not address dynamics and they don't venture hypotheses in this direction.

There is a relationship between the "distributed-plus-hub" view and [Dam00] better-known notion of a "convergence zone", defined roughly as a location where the brain binds features together. A convergence zone, in [Dam00] perspective, is not a "store" of information but an agent capable of decoding a signal (and of reconstructing information). He also uses the metaphor that convergence zones behave like indexes drawing information from other areas of the brain – but they are dynamic rather than static indices, containing the instructions needed to recognize and combine the features constituting the memory of something. The mechanism involved in the distributed-plus-hub model is similar to a convergence zone, but with the important difference that hubs are less local: [PNR07] semantic hub may be thought of a kind of

"cluster of convergence zones" consisting of a network of convergence zones for various semantic memories.

What is missing in [PNR07] and [Dam00] perspective is a vision of distributed memories as attractors. The idea of localized memories serving as indices into distributed knowledge stores is important, but is only half the picture of glocal memory: the creative, constructive, dynamical-attractor aspect of the distributed representation is the other half. The closest thing to a clear depiction of this aspect of glocal memory that seems to exist in the neuroscience literature is a portion of William Calvin's theory of the "cerebral code" [Cal96]. Calvin proposes a set of quite specific mechanisms by which knowledge may be represented in the brain using complexly-structured strange attractors, and by which these strange attractors may be propagated throughout the brain. Figure 13.2 shows one aspect of his theory: how a distributed attractor may propagate from one part of the brain to another in pieces, with one portion of the attractor getting propagated first, and then seeding the formation in the destination brain region of a close approximation of the whole attractor.

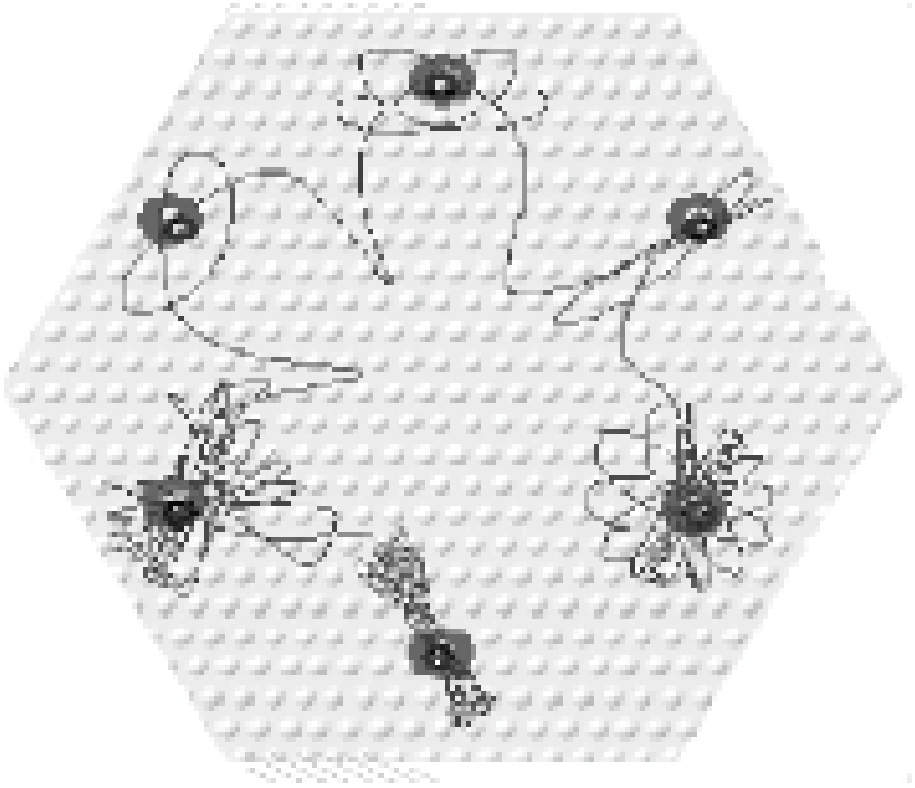


Fig. 13.2 Calvin's Model of Distributed Attractors in the Brain

Calvin's theory may be considered a genuinely glocal theory of memory. However, it also makes a large number of other specific commitments that are not part of the notion of glocality, such as his proposal of hexagonal metacolumns in the cortex, and his commitment to evolutionary learning as the primary driver of neural knowledge creation. We find these other hypotheses interesting and highly promising, yet feel it is also important to separate out the notion of glocal memory for separate consideration.

Regarding specifics, our suggestion is that Calvin's approach may overemphasize the distributed aspect of memory, not giving sufficient due to the relatively localized aspect as accounted for in the [QKKF08] results discussed above. In Calvin's glocal approach, global memories are attractors and local memories are parts of attractors. We suggest a possible alternative, in which global memories are attractors and local memories are particular neuronal subnetworks such as the specialized ones identified by [QKKF08]. However, this alternative does not seem contradictory to Calvin's overall conceptual approach, even though it is different from the particular proposals made in [Cal96].

The above paragraphs are far from a complete survey of the relevant neuroscience literature; there are literally dozens of studies one could survey pointing toward the glocality of various sorts of human memory. Yet experimental neuroscience tools are still relatively primitive, and every one of these studies could be interpreted in various other ways. In the next couple decades, as neuroscience tools improve in accuracy, our understanding of the role of glocality in human memory will doubtless improve tremendously.

13.6.3 Glocal Hopfield Networks

The ideas in the previous section suggest that, if one wishes to construct an AGI, it is worth seriously considering using a memory with some sort of glocal structure. One research direction that follows naturally from this notion is "glocal neural networks." In order to explore the nature of glocal neural networks in a relatively simple and tractable setting, we have formalized and implemented simple examples of "glocal Hopfield networks": palimpsest Hopfield nets with the addition of neurons representing localized memories. While these specific networks are not used in CogPrime

, they are quite similar to the ECAN networks that are used in CogPrime and described in Chapter 23 of Part 2.

Essentially, we augment the standard Hopfield net architecture by adding a set of "key neurons." These are a small percentage of the neurons in the network, and are intended to be roughly equinumerous to the number of memories the network is supposed to store. When the Hopfield net converges to an attractor A, then new links are created between the neurons that are active in A, and one of the key neurons. Which key neuron is chosen? The

one that, when it is stimulated, gives rise to an attractor pattern maximally similar to A.

The ultimate result of this is that, in addition to the distributed memory of attractors in the Hopfield net, one has a set of key neurons that in effect index the attractors. Each attractor corresponds to a single key neuron. In the glocal memory model, the key neurons are the keys and the Hopfield net attractors are the maps.

This algorithm has been tested in sparse Hopfield nets, using both standard Hopfield net learning rules and Storkey's modified palimpsest learning rule [SV99], which provides greater memory capacity in a continuous learning context. The use of key neurons turns out to slightly increase Hopfield net memory capacity, but this isn't the main point. The main point is that one now has a local representation of each global memory, so that if one wants to create a link between the memory and something else, it's extremely easy to do so – one just needs to link to the corresponding key neuron. Or, rather, one of the corresponding key neurons: depending on how many key neurons are allocated, one might end up with a number of key neurons corresponding to each memory, not just one.

In order to transform a palimpsest Hopfield net into a glocal Hopfield net, the following steps are taken:

1. Add a fixed number of "key neurons" to the network (removing other random neurons to keep the total number of neurons constant)
2. When the network reaches an attractor, create links from the elements in the attractor to one of the key neurons
3. The key neuron chosen for the previous step is the one that most closely matches the current attractor (which may be determined in several ways, to be discussed below)
4. To avoid the increase of the number of links in the network, when new links are created in Step 2, other key-neuron links are then deleted (several approaches may be taken here, but the simplest is to remove the key-neuron links with the lowest-absolute-value weights)

In the simple implementation of the above steps that we implemented, and described in [GPI+10], Step 3 is carried out simply by comparing the weights of a key neuron's links to the nodes in an attractor. A more sophisticated approach would be to select the key neuron with the highest activation during the transient interval immediately prior to convergence to the attractor.

The result of these modifications to the ordinary Hopfield net, is a Hopfield net that continually maintains a set of key neurons, each of which individually represents a certain attractor of the net.

Note that these key neurons – in spite of being "symbolic" in nature – are learned rather than preprogrammed, and are every bit as adaptive as the attractors they correspond to. Furthermore, if a key neuron is removed, the glocal Hopfield net algorithm will eventually learn it back, so the robustness properties of Hopfield nets are retained.

The results of experimenting with glocal Hopfield nets of this nature are summarized in [GPI+10]. We studied Hopfield nets with connectivity around .1, and in this context we found that glocality

- slightly increased memory capacity
- massively increased the rate of convergence to the attractor, i.e. the speed of recall

However, probably the most important consequence of glocality is a more qualitative one: it makes it far easier to link the Hopfield net into a larger system, as would occur if the Hopfield net were embedded in an integrative AGI architecture. Because a neuron external to the Hopfield net may now link to a memory in the Hopfield net by linking to the corresponding key neuron.

13.6.4 Neural-Symbolic Glocality in CogPrime

In CogPrime , we have explicitly sought to span the symbolic/emergentist pseudo-dichotomy, via creating an integrative knowledge representation that combines logic-based aspects with neural-net-like aspects. As reviewed in Chapter 6 above, these function not in the manner of multimodular systems, but rather via using (probabilistic) truth values and (attractor neural net like) attention values as weights on nodes and links of the same (hyper) graph. The nodes and links in this hypergraph are typed, like a standard semantic network approach for knowledge representation, so they're able to handle all sorts of knowledge, from the most concrete perception and actuation related knowledge to the most abstract relationships. But they're also weighted with values similar to neural net weights, and pass around quantities (importance values, discussed in Chapter 23 of Part 2) similar to neural net activations, allowing emergent attractor/assembly based knowledge representation similar to attractor neural nets.

The concept of glocality lies at the heart of this combination, in a way that spans the pseudo-dichotomy:

- Local knowledge is represented in abstract logical relationships stored in explicit logical form, and also in Hebbian-type associations between nodes and links.
- Global knowledge is represented in large-scale patterns of node and link weights, which lead to large-scale patterns of network activity, which often take the form of attractors qualitatively similar to Hopfield net attractors. These attractors are called *maps*.

The result of all this is that a concept like "cat" might be represented as a combination of:

- A small number of logical relationships and strong associations, that constitute the "key" subnetwork for the "cat" concept.
- A large network of weak associations, binding together various nodes and links of various types and various levels of abstraction, representing the "cat map".

The activation of the key will generally cause the activation of the map, and the activation of a significant percentage of the map will cause the activation of the rest of the map, including the key. Furthermore, if the key were for some reason forgotten, then after a significant amount of effort, the system would likely to be able to reconstitute it (perhaps with various small changes) from the information in the map. We conjecture that this particular kind of glocal memory will turn out to be very powerful for AGI, due to its ability to combine the strengths of formal logical inference with those of self-organizing attractor neural networks.

As a simple example, consider the representation of a "tower", in the context of an artificial agent that has built towers of blocks, and seen pictures of many other kinds of towers, and seen some tall building that it knows are somewhat like towers but perhaps not exactly towers. If this agent is reasonably conceptually advanced (say, at Piagetan the concrete operational level) then its mind will contain some declarative relationships partially characterizing the concept of "tower," as well as its sensory and episodic examples, and its procedural knowledge about how to build towers.

The key of the "tower" concept in the agent's mind may consist of internal images and episodes regarding the towers it knows best, the essential operations it knows are useful for building towers (piling blocks atop blocks atop blocks...), and the core declarative relations summarizing "towerness" – and the whole "tower" map then consists of a much larger number of images, episodes, procedures and declarative relationships connected to "tower" and other related entities. If any portion of the map is removed – even if the key is removed – then the rest of the map can be approximately reconstituted, after some work. Some cognitive operations are best done on the localized representation – e.g. logical reasoning. Other operations, such as attention allocation and guidance of inference control, are best done using the globalized map representation.

Chapter 14
Representing Implicit Knowledge via Hypergraphs

14.1 Introduction

Explicit knowledge is easy to write about and talk about; implicit knowledge is equally important, but tends to get less attention in discussions of AI and psychology, simply because we don't have as good a vocabulary for describing it, nor as good a collection of methods for measuring it. One way to deal with this problem is to describe implicit knowledge using language and methods typically reserved for explicit knowledge. This might seem intrinsically non-workable, but we argue that it actually makes a lot of sense. The same sort of networks that a system like CogPrime

uses to represent knowledge explicitly, can also be used to represent the *emergent* knowledge that implicitly exists in an intelligent system's complex structures and dynamics.

We've noted that CogPrime uses an explicit representation of knowledge in terms of weighted labeled hypergraphs; and also uses other more neural net like mechanisms (e.g. the economic attention allocation network subsystem) to represent knowledge globally and implicitly. CogPrime combines these two sorts of representation according to the principle we have called *glocality*. In this chapter we pursue glocality a bit further – describing a means by which even implicitly represented knowledge can be modeled using weighted labeled hypergraphs similar to the ones used explicitly in CogPrime . This is conceptually important, in terms of making clear the fundamental similarities and differences between implicit and explicit knowledge representation; and it is also pragmatically meaningful due to its relevance to the CogPrime methods described in Chapter 42 of Part 2 that transform implicit into explicit knowledge.

To avoid confusion with CogPrime 's explicit knowledge representation, we will refer to the hypergraphs in this chapter as composed of Vertices and Edges rather than Nodes and Links. In prior publications we have referred to "derived" or "emergent" hypergraphs of the sort described here using the

acronym **SMEPH**, which stands for Self-Modifying, Evolving Probabilistic Hypergraphs.

14.2 Key Vertex and Edge Types

We begin by introducing a particular collection of Vertex and Edge types, to be used in modeling the internal structures of intelligent systems.

The key SMEPH Vertex types are

- ConceptVertex, representing a set, for instance, an idea or a set of percepts
- SchemaVertex, representing a procedure for doing something (perhaps something in the physical world, or perhaps an abstract mental action).

The key SMEPH Edge types, using language drawn from Probabilistic Logic Networks (PLN) and elaborated in Chapter 34 below, are as follows:

- ExtensionalInheritanceEdge (ExtInhEdge for short: an edge which, linking one Vertex or Edge to another, indicates that the former is a special case of the latter)
- ExtensionalSimilarityEdge (ExtSim: which indicates that one Vertex or Edge is similar to another)
- ExecutionEdge (a ternary edge, which joins S,B,C when S is a SchemaVertex and the result from applying S to B is C).

So, in a SMEPH system, one is often looking at hypergraphs whose Vertices represent ideas or procedures, and whose Edges represent relationships of specialization, similarity or transformation among ideas and/or procedures.

The semantics of the SMEPH edge types is given by PLN, but is simple and commonsensical. ExtInh and ExtSim Edges come with probabilistic weights indicating the extent of the relationship they denote (e.g. the ExtSimEdge joining the cat ConceptVertex to the dog ConceptVertex gets a higher probability weight than the one joining the cat ConceptVertex to the washing-machine ConceptVertex). The mathematics of transformations involving these probabilistic weights becomes quite involved $\tilde{N}$ particularly when one introduces SchemaVertices corresponding to abstract mathematical operations, a step that enables SMEPH hypergraphs to have the complete mathematical power of standard logical formalisms like predicate calculus, but with the added advantage of a natural representation of uncertainty in terms of probabilities, as well as a natural representation of networks and webs of complex knowledge.

14.3 Derived Hypergraphs

We now describe how SMEPH hypergraphs may be used to model and describe intelligent systems. One can (in principle) draw a SMEPH hypergraph corresponding to any individual intelligent system, with Vertices and Edges for the concepts and processes in that system's mind. This is called the derived hypergraph of that system.

14.3.1 SMEPH Vertices

A ConceptVertex in the derived hypergraph of a system corresponds to a structural pattern that persists over time in that system; whereas a SchemaVertex corresponds to a multi-time-point dynamical pattern that recurs in that system's dynamics. If one accepts the patternist definition of a mind as the set of patterns in an intelligent system, then it follows that the derived hypergraph of an intelligent system captures a significant fraction of the mind of that system.

To phrase it a little differently, we may say that a ConceptVertex, in SMEPH, refers to the habitual pattern of activity observed in a system when some condition is met (this condition corresponding to the presence of a certain pattern). The condition may refer to something in the world external to the system, or to something internal. For instance, the condition may be observing a cat. In this case, the corresponding Concept vertex in the mind of Ben Goertzel is the pattern of activity observed in Ben Goertzel's brain when his eyes are open and he's looking in the direction of a cat. The notion of pattern of activity can be made rigorous using mathematical pattern theory, as is described in The Hidden Pattern [Goe06a].

Note that logical predicates, on the SMEPH level, appear as particular kinds of Concepts, where the condition involves a predicate and an argument. For instance, suppose one wants to know what happens inside Ben's mind when he eats cheese. Then there is a Concept corresponding to the condition of cheese-eating activity. But there may also be a Concept corresponding to eating activity in general. If the Concept denoting the activity of eating X is generally easily computable from the Concepts for X and eating individually, then the eating Concept is effectively acting as a predicate.

A SMEPH SchemaVertex, on the other hand, is like a Concept that's defined in a time-dependent way. One type of Schema refers to a habitual dynamical pattern of activity occurring before and/or during some condition is met. For instance, the condition might be saying the word Hello. In that case the corresponding SchemaVertex in the mind of Ben Goertzel is the pattern of activity that generally occurs before he says Hello.

Another type of Schema refers to a habitual dynamical pattern of activity occurring after some condition X is met. For instance, in the case of the

Schema for adding two numbers, the precondition X consists of the two numbers and the concept of addition. The Schema is then what happens when the mind thinks of adding and thinks of two numbers.

Finally, there are Schema that refer to habitual dynamical activity patterns occurring after some condition X is met and before some condition Y is met. In this case the Schema is viewed as transforming X into Y. For instance, if X is the condition of meeting someone who is not a friend, and Y is the condition of being friends with that person, then the habitually intervening activities constitute the Schema for making friends.

14.3.2 SMEPH Edges

SMEPH edge types fall into two categories: functional and logical. Functional edges connect Schema vertices to their input and outputs; logical edges refer mainly to conditional probabilities, and in general are to be interpreted according to the semantics of Probabilistic Logic Networks.

Let us begin with logical edges. The simplest case is the Subset edge, which denotes a straightforward, extensional conditional probability. For instance, it may happen that whenever the Concept for cat is present in a system, the Concept for animal is as well. Then we would say

```
Subset cat animal
```

(Here we assume a notation where "R A B" denotes an Edge of type R between Vertices A and B.)

On the other hand, it may be that 50% of the time that cat is present in the system, cute is present as well: then we would say

```
Subset cat cute <.5>
```

where the <.5> denotes the probability, which is a component of the Truth Value associated with the edge.

Next, the most basic functional edge is the Execution edge, which is ternary and denotes a relation between a Schema, its input and its output, e.g.

```
Execution father_of Ben_Goertzel Ted_Goertzel
```

for a schema father_of that outputs the father of its argument.

The ExecutionOutput (ExOut) edge denotes the output of a Schema in an implicit way, e.g.

```
ExOut say_hello
```

refers to a particular act of saying hello, whereas

```
ExOut add_numbers {3, 4)
```

refers to the Concept corresponding to 7. Note that this latter example involves a set of three entities: sets are also part of the basic SMEPH knowledge

representation. A set may be thought of as a hypergraph edge that points to all its members.

In this manner we may define a set of edges and vertices modeling the habitual activity patterns of a system when in different situations. This is called the derived hypergraph of the system. Note that this hypergraph can in principle be constructed no matter what happens inside the system: whether it's a human brain, a formal neural network, Cyc, OCP, a quantum computer, etc. Of course, constructing the hypergraph in practice is quite a different story: for instance, we currently have no accurate way of measuring the habitual activity patterns inside the human brain. fMRI and PET and other neuroimaging technologies give only a crude view, though they are continually improving.

Pattern theory enters more deeply here when one thoroughly fleshes out the Inheritance concept. Philosophers of logic have extensively debated the relationship between extensional inheritance (inheritance between sets based on their members) and intensional inheritance (inheritance between entity-types based on their properties). A variety of formal mechanisms have been proposed to capture this conceptual distinction; see (Wang, 2006, 1995 TODO make ref) for a review along with a novel approach utilizing uncertain term logic. Pattern theory provides a novel approach to defining intension: one may associate with each ConceptVertex in a system's derived hypergraph the set of patterns associated with the structural pattern underlying that ConceptVertex. Then, one can define the strength of the IntensionalInheritanceEdge between two ConceptVertices A and B as the percentage of A's pattern-set that is also contained in B's pattern-set. According to this approach, for instance, one could have

```
IntInhEdge whale fish <0.6>

ExtInhEdge whale fish <0.0>
```

since the fish and whale sets have common properties but no common members.

14.4 Implications of Patternist Philosophy for Derived Hypergraphs of Intelligent Systems

Patternist philosophy rears its head here and makes some definite hypotheses about the structure of derived hypergraphs. It suggests that derived hypergraphs should have a dual network structure, and that in highly intelligent systems they should have subgraphs that constitute models of the whole hypergraph (these are self systems). SMEPH does not add anything to the patternist view on a philosophical level, but it gives a concrete instantiation to some of the general ideas of patternism. In this section we'll articulate

some "SMEPH principles", constituting important ideas from patternist philosophy as they manifest themselves in the SMEPH context.

The logical edges in a SMEPH hypergraph are weighted with probabilities, as in the simple example given above. The functional edges may be probabilistically weighted as well, since some Schema may give certain results only some of the time. These probabilities are critical in terms of SMEPH's model of system dynamics; they underly one of our SMEPH principles,

Principle of Implicit Probabilistic Inference: In an intelligent system, the temporal evolution of the probabilities on the edges in the system's derived hypergraph should approximately obey the rules of probability theory.

The basic idea is that, even if a system's underlying dynamics has no explicit connection to probability theory, nevertheless it must behave roughly as if it does, if it is going to be intelligent. The roughly part is important here; it's well known that humans are not terribly accurate in explicitly carrying out formal probabilistic inferences. And yet, in practical contexts where they have experience, humans can make quite accurate judgments; which is all that's required by the above principle, since it's the contexts where experience has occurred that will make up a system's derived hypergraph.

Our next SMEPH principle is evolutionary, and states

Principle of Implicit Evolution: In an intelligent system, new Schema and Concepts will continually be created, and the Schema and Concepts that are more useful for achieving system goals (as demonstrated via probabilistic implication of goal achievement) will tend to survive longer.

Note that this principle can be fulfilled in many different ways. The important thing is that system goals are allowed to serve as a selective force.

Another SMEPH dynamical principle pertains to a shorter time-scale than evolution, and states

Principle of Attention Allocation: In an intelligent system, Schema and Concepts that are more useful for attaining short-term goals will tend to consume more of the system's energy. (The balance of attention oriented toward goals pertaining to different time scales will vary from system to system.)

Next, there is the

Principle of Autopoesis: In an intelligent system, if one removes some part of the system and then allows the system's natural dynamics to keep going, a decent approximation to that removed part will often be spontaneously reconstituted.

And the

Cognitive Equation Principle: In an intelligent system, many abstract patterns that are present in the system at a certain time as patterns among other Schema and Concepts, will at a near-future time be present in the system as patterns among elementary system components.

The Cognitive Equation Principle, briefly discussed in Chapter 3 abovee, basically means that Concepts and Schema emergent in the system are recognized by the system and then embodied as elementary items in the system so that patterns among them in their emergent form become, with the passage of time, patterns among them in their directly-system-embodied form. This is a natural consequence of the way intelligent systems continually recognize patterns in themselves.

Note that derived hypergraphs may be constructed corresponding to any complex system which demonstrates a variety of internal dynamical patterns depending on its situation. However, if a system is not intelligent, then according to the patternist philosophy evolution of its derived hypergraph can't necessarily be expected to follow the above principles.

14.4.1 SMEPH Principles in CogPrime

We now more explicitly elaborate the application of these ideas in the CogPrime context. As noted above, in addition to explicit knowledge representation in terms of Nodes and Links, CogPrime also incorporates implicit knowledge representation in the form of what are called Maps: collections of Nodes and Links that tend to be utilized together within cognitive processes.

These Maps constitute a CogPrime system's derived hypergraph, which will not be identical to the hypergraph it uses for explicit knowledge representation. However, an interesting feedback loop arises here, in that the intelligence's self-study will generally lead it to recognize large portions of its derived hypergraph as patterns in itself, and then embody these patterns within its concretely implemented knowledge hypergraph. This is closely related to the Cognitive Equation phenomenon described in Chapter 3, in which an intelligent system continually recognizes patterns in itself and embodies these patterns in its own basic structure (so that new patterns may more easily emerge from them).

Often it happens that a particular CogPrime node will serve as the center of a map, so that e.g. the Concept Link denoting cat will consist of a number of nodes and links roughly centered around a ConceptNode that is linked to the WordNode cat. But this is not guaranteed and some CogPrime maps are more diffuse than this with no particular center.

Somewhat similarly, the key SMEPH dynamics are represented explicitly in CogPrime : probabilistic reasoning is carried out via explicit application of PLN on the CogPrime hypergraph, evolutionary learning is carried out via application of the MOSES optimization algorithm, and attention allocation is carried out via a combination of inference and evolutionary pattern mining. But the SMEPH dynamics also occur implicitly in CogPrime : emergent maps are reasoned on probabilistically as an indirect consequence of node-and-link level PLN activity; maps evolve as a consequence of the coordinated whole of CogPrime dynamics; and attention shifts between maps according to complex emergent dynamics.

To see the need for maps, consider that even a Node that has a particular meaning attached to it - like the *Iraq* Node, say - doesn't contain much of the meaning of *Iraq* in it. The meaning of *Iraq* lies in the Links attached to this Node, and the Links attached to their Nodes - and the other Nodes and Links not explicitly represented in the system, which will be created by CogPrime 's cognitive algorithms based on the explicitly existent Nodes and Links related to the *Iraq* Node.

This halo of Atoms related to the *Iraq* node is called the *Iraq* map. In general, some maps will center around a particular Atom, like this *Iraq* map, others may not have any particular identifiable center. CogPrime 's cognitive processes act directly on the level of Nodes and Links, but they must be analyzed in terms of their impact on maps as well. In SMEPH terms, CogPrime maps may be said to correspond to SMEPH ConceptNodes, and for instance bundles of Links between the Nodes belonging to a map may correspond to a SMEPH Link between two ConceptNodes

Chapter 15
Emergent Networks of Intelligence

15.1 Introduction

When one is involved with engineering an AGI system, one thinks a lot about the aspects of the system one is explicitly building – what are the parts, how they fit together, how to test they're properly working, and so forth. And yet, these explicitly engineered aspects are only a fraction of what's important in an AGI system. At least as critical are the *emergent* aspects – the patterns that emerge once the system is up and running, interacting with the world and other agents, growing and developing and learning and self-modifying. SMEPH is one toolkit for describing some of these emergent patterns, but it's only a start.

In line with these general observations, most of this book we will focus on the structures and processes that we have built, or intend to build, into the CogPrime system. But in a sense, these structures and processes are not the crux of CogPrime 's intended intelligence. The purpose of these preprogrammed structures and processes is to give rise to *emergent* structures and processes, in the course of CogPrime 's interaction with the world and the other minds within it. We will return to this theme of emergence at several points in later chapters, e.g. in the discussion of map formation in Chapter 42 of Part 2.

Given the important of emergent structures – and specifically emergent *network* structures – for intelligence, it's fortunate the scientific community has already generated a lot of knowledge about complex networks: both networks of physical or software elements, and networks of organization emergent from complex systems. As most of this knowledge has originated in fields other than AGI, or in pure mathematics, it tends to require some reinterpretation or tweaking to achieve maximal applicability in the AGI context; but we believe this effort will become increasingly worthwhile as the AGI field progresses, because network theory is likely to be very useful for describ-

ing the contents and interactions of AGI systems as they develop increasing intelligence.

In this brief chapter we specifically focus on the emergence of certain large-scale *network structures* in a CogPrime knowledge store, presenting heuristic arguments as to why these structures can be expected to arise. We also comment on the way in which these emergent structures are expected to guide cognitive processes, and give rise to emergent cognitive processes. The following chapter expands on this theme in a particular direction, exploring the possible emergence of structures characterizing inter-cognitive reflection.

15.2 Small World Networks

One simple but potentially useful observation about CogPrime Atomspaces is that they are generally going to be *small world networks* [**?**], rather than random graphs. A small world network is a graph in which the connectivities of the various nodes display a power law behavior – so that, loosely speaking, there are a few nodes with very many links, then more nodes with a modest number of links ... and finally, a huge number of nodes with very few links. This kind of network occurs in many natural and human systems, including citations among papers, financial arrangements among banks, links between Web pages and the spread of diseases among people or animals. In a weighted network like an Atomspace, "small-world-ness" must be defined in a manner taking the weights into account, and there are several obvious ways to do this. Figure 15.1 depicts a small but prototypical small-worlds network, with a few "hub" nodes possessing far more neighbors than the others, and then some secondary hubs, etc.

An excellent reference on network theory in general, including but not limited to small world networks, is Peter Csermely's *Weak Links* [**?**]. Many of the ideas in that work have apparent OpenCog applications, which are not elaborated here.

One process via which small world networks commonly form is "preferential attachment" [**?**]. This occurs in essence when "the rich get richer" – i.e. when nodes in the network grow new links, in a manner that causes them to preferentially grow links to nodes that already have more links. It is not hard to see that CogPrime 's ECAN dynamics will naturally lead to preferential attachment, because Atoms with more links will tend to get more STI, and thus will tend to get selected by more cognitive processes, which will cause them to grow more links. For this reason, in most circumstances, a CogPrime system in which most link-building cognitive processes rely heavily on ECAN to guide their activities will tend to contain a small-world-network Atomspace. This is not rigorously guaranteed to be the case for any possible combination of environment and goals, but it is commonsensically likely to nearly always be the case.

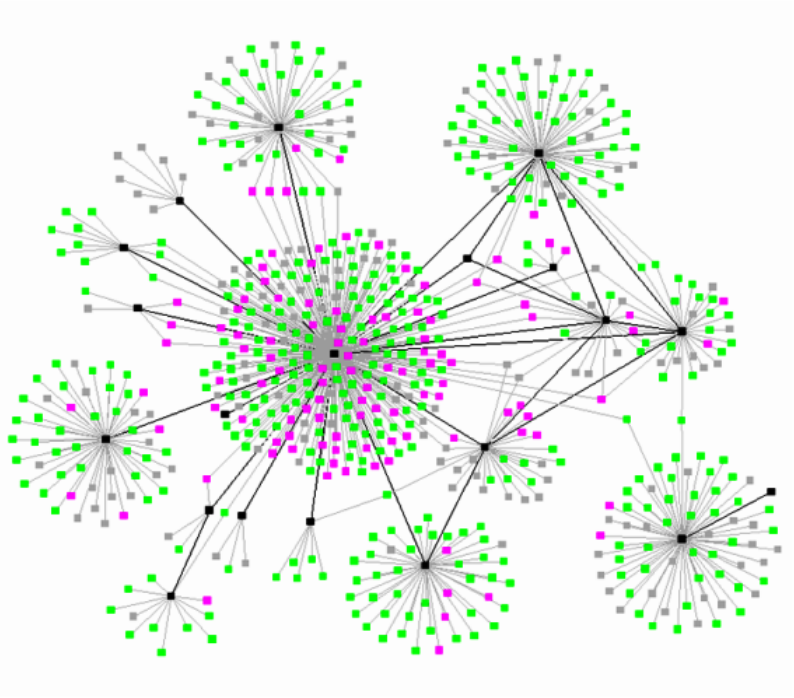


Fig. 15.1 A typical, though small-sized, small-worlds network.

One consequence of the small worlds structure of the Atomspace is that, in exploring other properties of the Atom network, it is particularly important to look at the hub nodes. For instance, if one is studying whether hierarchical and heterarchical subnetworks of the Atomspace exist, and whether they are well-aligned with each other, it is important to look at hierarchical and heterarchical connections between hub nodes in particular (and secondary hubs, etc.). A pattern of hierarchical or dual network connection that only held up among the more sparsely connected nodes in a small-world network would be a strange thing, and perhaps not that cognitively useful.

15.3 Dual Network Structure

One of the key theoretical notions in patternist philosophy is that complex cognitive systems evolve internal *dual network* structures, comprising superposed, harmonized hierarchical and heterarchical networks. Now we explore some of the specific CogPrime structures and dynamics militating in favor of the emergence of dual networks.

15.3.1 Hierarchical Networks

The hierarchical nature of human linguistic concepts is well known, and is illustrated in Figure 15.2 for the commonsense knowledge domain (using a graph drawn from WordNet, a huge concept hierarchy covering 50K+ English-language concepts), and in Figure 15.4 for a specialized knowledge subdomain, genetics. Due to this fact, a certain amount of hierarchy can be expected to emerge in the Atomspace of any linguistically savvy CogPrime , simply due to its modeling of the linguistic concepts that it hears and reads.

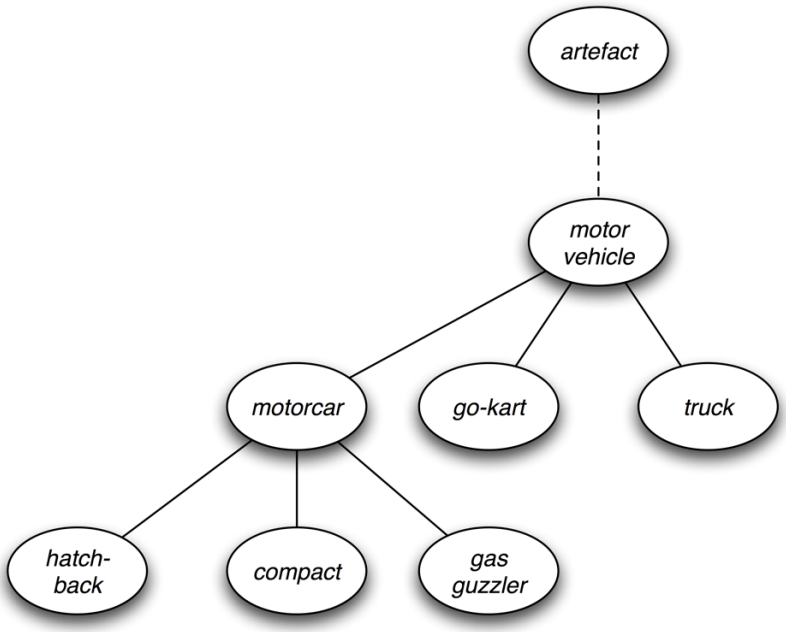


Fig. 15.2 A typical, though small, subnetwork of WordNet's hierarchical network.

Hierarchy also exists in the natural world apart from language, which is the reason that many sensorimotor-knowledge-focused AGI systems (e.g. DeSTIN and HTM, mentioned in Chapter 4 above) feature hierarchical structures. In these cases the hierarchies are normally spatiotemporal in nature, with lower layers containing elements responding to more localized aspects of the perceptual field, and smaller, more localized groups of actuators. This kind of hierarchy certainly *could* emerge in an AGI system, but in CogPrime we have opted for a different route. If a CogPrime system is hybridized with a hierarchical sensorimotor network like one of those mentioned above, then the Atoms linked to the nodes in the hierarchical sensorimotor network will nat-

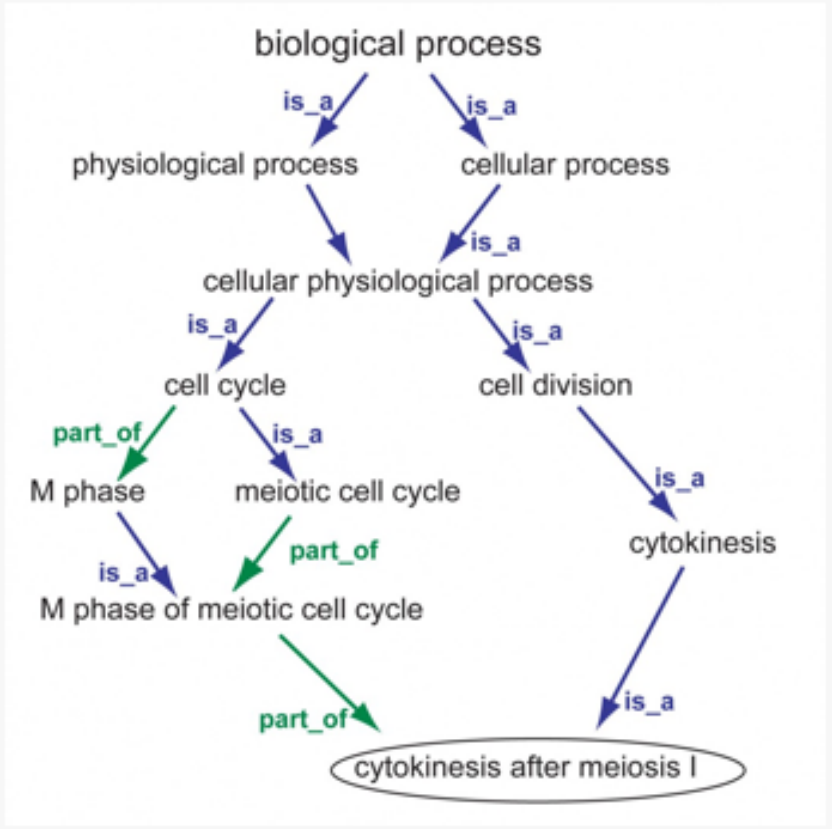


Fig. 15.3 A typical, though small, subnetwork of the Gene Ontology's hierarchical network.

urally possess hierarchical conceptual relationships, and will thus naturally grow hierarchical links between them (e.g. InheritanceLinks and IntensionalInheritanceLinks via PLN, AsymmetricHebbianLinks via ECAN).

Once elements of hierarchical structure exist via the hierarchical structure of language and physical reality, then a richer and broader hierarchy can be expected to accumulate on top of it, because importance spreading and inference control will implicitly and automatically be guided by the existing hierarchy. That is, in the language of *Chaotic Logic* [?] and patternist theory, hierarchical structure is an "autopoietic attractor" – once it's there it will tend to enrich itself and maintain itself. AsymmetricHebbianLinks arranged in a hierarchy will tend to cause importance to spread up or down the hierarchy, which will lead other cognitive processes to look for patterns between Atoms and their hierarchical parents or children, thus potentially building more hierarchical links. Chains of InheritanceLinks pointing up and down the hierarchy will lead PLN to search for more hierarchical links – e.g.

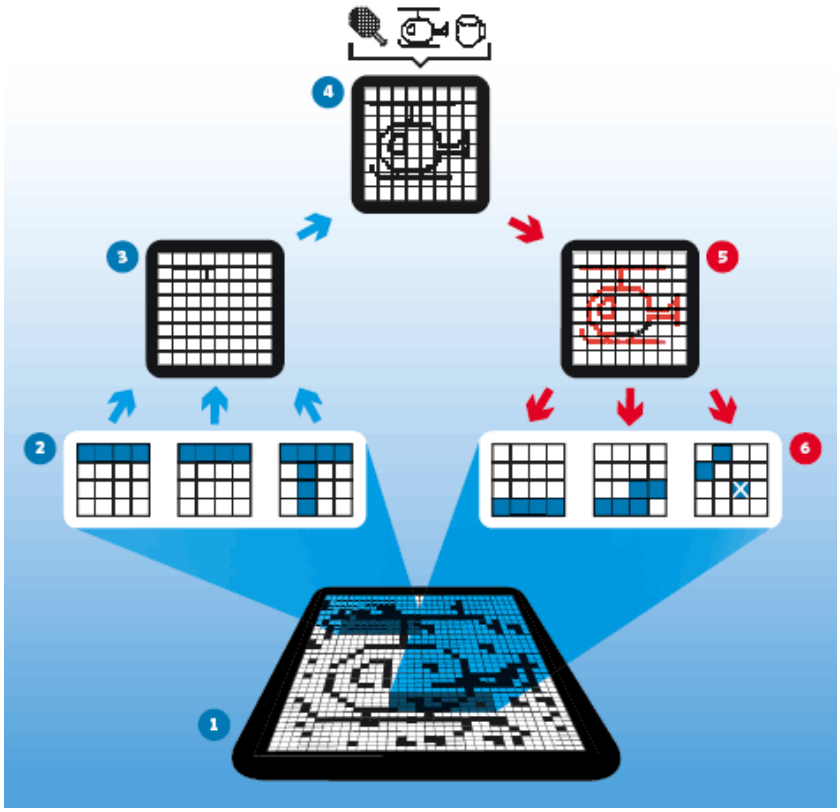


Fig. 15.4 Small-scale portrayal of a portion of the spatiotemporal hierarchy in Jeff Hawkins' Hierarchical Temporal Memory architecture.

most simply, $A \to B \to C$ where C is above B is above A in the hierarchy, will naturally lead inference to check the viability of $A \to C$ by deduction. There is also the possibility to introduce a special DefaultInheritanceLink, as discussed in Chapter 34 of Part 2, but this isn't actually necessary to obtain the inferential maintenance of a robust hierarchical network.

15.3.2 Associative, Heterarchical Networks

Heterarchy is in essence a simpler structure than hierarchy: it simply refers to a network in which nodes are linked to other nodes with which they share important relationships. That is, there should be a tendency that if two nodes are often important in the same contexts or for the same purposes, they

should be linked together. Portrayals of typical heterarchical linkage patterns among natural language concepts are given in Figures 15.5 and 15.6. Just for fun, Figure 15.7 shows one person's attempt to draw a heterarchical graph of the main concepts in one of Douglas Hofstadter's books. Naturally, real concept heterarchies are far more large, complex and tangled than even this one.

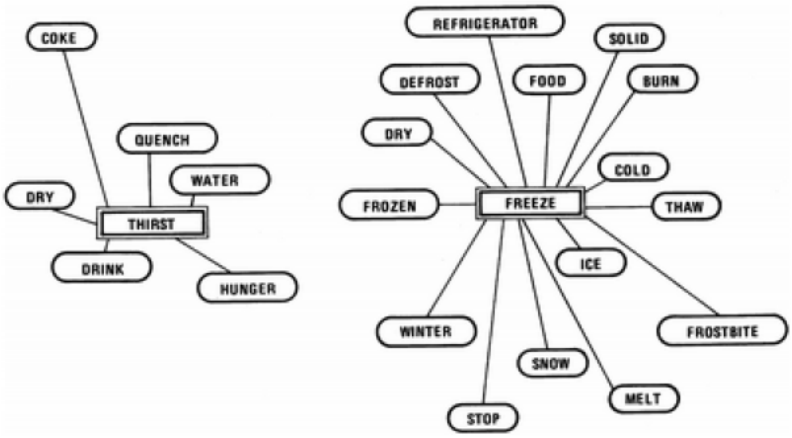


Fig. 15.5 Portions of a conceptual heterarchy centered on specific concepts.

In CogPrime , ECAN enforces heterarchy via building SymmetricHebbianLinks, and PLN by building SimilarityLinks, IntensionalSimilarityLinks and ExtensionalSimilarityLinks. Furthermore, these various link types reinforce each other. PLN control is guided by importance spreading, which follows Hebbian links, so that a heterarchical Hebbian network tends to cause PLN to explore the formation of links following the same paths as the heterarchical HebbianLinks. And importance can spread along logical links as well as explicit Hebbian links, so that the existence of a heterarchical logical network will tend to cause the formation of additional heterarchical Hebbian links. Heterarchy reinforces itself in "autopoietic attractor" style even more simply and directly than heterarchy.

15.3.3 Dual Networks

Finally, if both hierarchical and heterarchical structures exist in an Atomspace, then both ECAN and PLN will naturally blend them together, because hierarchical and heterarchical links will feed into their link-creation processes and naturally be combined together to form new links. This will tend to pro-

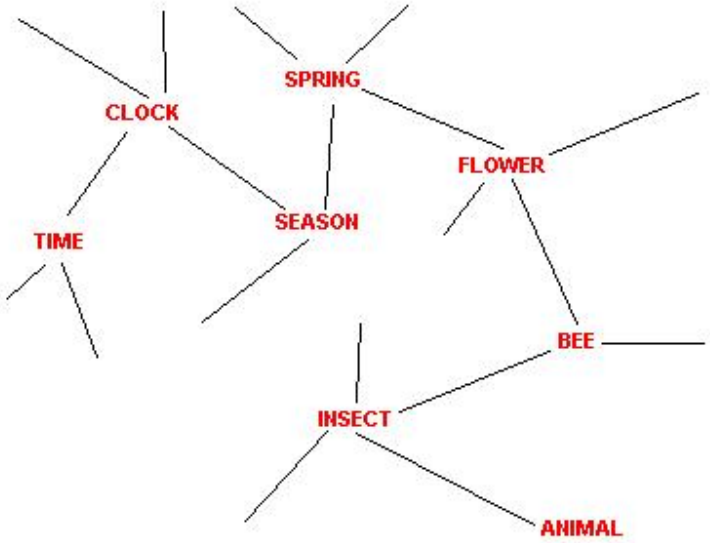


Fig. 15.6 A portion of a conceptual heterarchy, showing the "dangling links" leading this portion to the rest of the heterarchy.

duce a structure called a *dual network*, in which a hierarchy exists, along with a rich network of heterarchical links joining nodes in the hierarchy, with a particular density of links between nodes on the same hierarchical level. The dual network structure will emerge without any explicit engineering oriented toward it, simply via the existence of hierarchical and heterarchical networks, and the propensity of ECAN and PLN to be guided by both the hierarchical and heterarchical networks. The existence of a natural dual network structure in both linguistic and sensorimotor data will help the formation process along, and then creative cognition will enrich the dual network yet further than is directly necessitated by the external world.

A rigorous mathematical analysis of the formation of hierarchical, heterarchical and dual networks in CogPrime systems has not yet been undertaken, and would certainly be an interesting enterprise. Similar to the theory of small world networks, there is ample ground here for both theorem-proving and heuristic experimentation. However, the qualitative points made here are sufficiently well-grounded in intuition and experience to be of some use guiding our ongoing work. One of the nice things about emergent network structures is that they are relatively straightforward to observe in an evolving, learning AGI system, via visualization and inspection of structures such at the Atomspace.

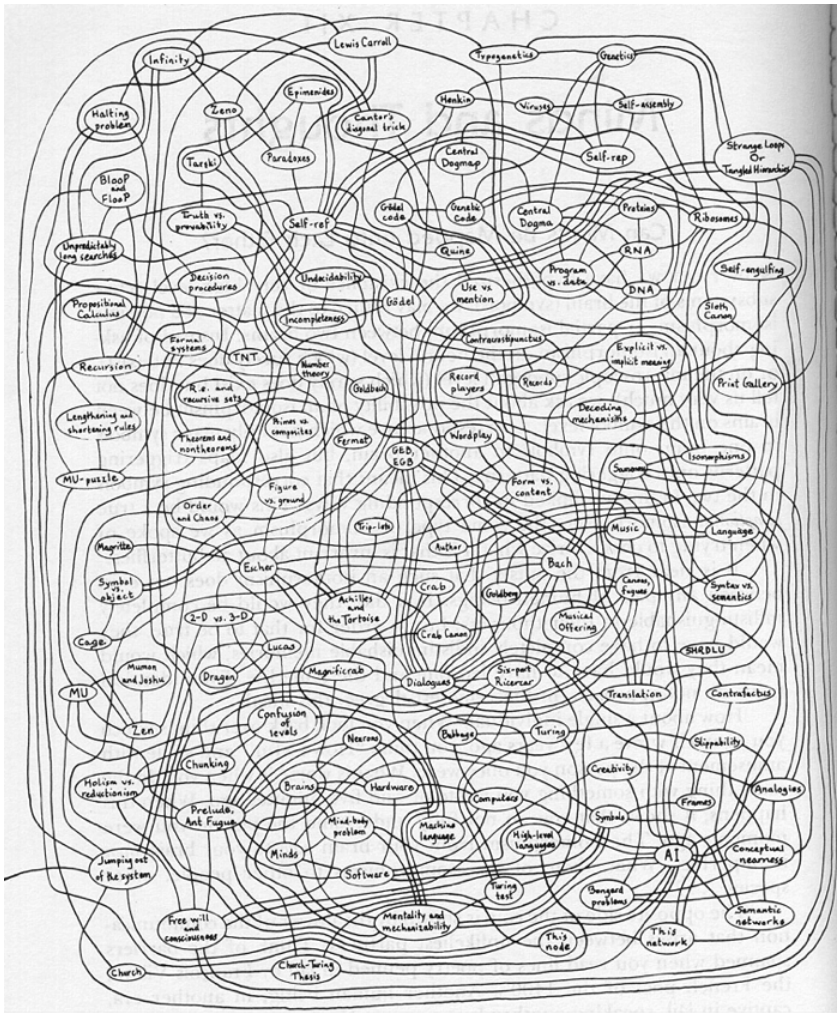


Fig. 15.7 A fanciful evocation of part of a reader's conceptual heterarchy related to Douglas Hofstadter's writings.

Section V
A Path to Human-Level AGI

Chapter 16
AGI Preschool

Co-authored with Stephan Vladimir Bugaj

16.1 Introduction

In conversations with government funding sources or narrow AI researchers about AGI work, one of the topics that comes up most often is that of "evaluation and metrics" – i.e., AGI intelligence testing. We actually prefer to separate this into two topics: environments and methods for careful qualitative evaluation of AGI systems, versus metrics for precise measurement of AGI systems. The difficulty of formulating bulletproof metrics for partial progress toward advanced AGI has become evident throughout the field, and in Chapter 8 we have elaborated one plausible explanation for this phenomenon, the "trickiness" of cognitive synergy. [LWML09], summarizing a workshop on "Evaluation and Metrics for Human-Level AI" held in 2008, discusses some of the general difficulties involved in this type of assessment, and some requirements that any viable approach must fulfill. On the other hand, the lack of appropriate methods for careful qualitative evaluation of AGI systems has been much less discussed, but we consider it actually a more important issue – as well as an easier (though not easy) one to solve.

We haven't actually found the lack of quantitative intelligence metrics to be a major obstacle in our practical AGI work so far. Our OpenCogPrime implementation lags far behind the CogPrime design as articulated in Part 2 of this book, and according to the theory underlying CogPrime , the more interesting behaviors and dynamics of the system will occur only when all the parts of the system have been engineered to a reasonable level of completion and integrated together. So, the lack of a great set of metrics for evaluating the intelligence of our partially-built system hasn't impaired too much. Testing the intelligence of the current OpenCogPrime system is a bit like testing the flight capability of a partly-built airplane that only has stubs for wings, lacks tail-fins, has a much less efficient engine than the one that's been designed for use in the first "real" version of the airplane, etc. There may be something to be learned from such preliminary tests, but making them highly rigorous

isn't a great use of effort, compared to working on finishing implementing the design according to the underlying theory.

On the other hand, the problem of what environments and methods to use to qualitatively evaluate and study AGI progress, has been considerably more vexing to us in practice, as we've proceeded in our work on implementing and testing OpenCogPrime and developing the CogPrime theory. When developing a complex system, it's nearly always valuable to see what this system does in some fairly rich, complex situations, in order to gain a better intuitive understanding of the parts and how they work together. In the context of human-level AGI, the theoretically best way to do this would be to embody one's AGI system in a humanlike body and set it loose in the everyday human world; but of course, this isn't feasible given the current state of development of robotics technology. So one must seek approximations. Toward this end we have embodied OpenCogPrime in non-player characters in video game style virtual worlds, and carried out preliminary experiments embodying OpenCogPrime in humanoid robots. These are reasonably good options but they have limitations and lead to subtle choices: what kind of game characters and game worlds, what kind of robot environments, etc.?

One conclusion we have come to, based largely on the considerations in Chapter 11 on development and Chapter 9 on the importance of environment, is that it may make sense to embed early-stage proto-AGI and AGI systems in environments reminiscent of those used for teaching young human children. In this chapter we will explore this approach in some detail: emulation, in either physical reality or an multiuser online virtual world, of an environment similar to preschools used in early human childhood education. Complete specification of an "AGI Preschool" would require much more than a brief chapter; our goal here is to sketch the idea in broad outline, and give a few examples of the types of opportunities such an environment would afford for instruction, spontaneous learning and formal and informal evaluation of certain sorts of early-stage AGI systems.

The material in this chapter will pop up fairly often later in the book. The AGI Preschool context will serve, throughout the following chapters, as a source of concrete examples of the various algorithms and structures. But it's not proposed merely as an expository tool; we are making the very serious proposal that sending AGI systems to a virtual or robotic preschool is an excellent way – perhaps the best way – to foster the development of human-level human-like AGI.

16.1.1 Contrast to Standard AI Evaluation Methodologies

The reader steeped in the current AI literature may wonder why it's necessary to introduce a new methodology and environment for evaluating AGI systems.

There are already very many different ways of evaluating AI systems out there ... do we really need another?

Certainly, the AI field has inspired many competitions, each of which tests some particular type or aspect of intelligent behavior. Examples include robot competitions, tournaments of computer chess, poker, backgammon and so forth at computer olympiads, trading-agent competition, language and reasoning competitions like the Pascal Textual Entailment Challenge, and so on. In addition to these, there are many standard domains and problems used in the AI literature that are meant to capture the essential difficulties in a certain class of learning problems: standard datasets for face recognition, text parsing, supervised classification, theorem-proving, question-answering and so forth.

However, the value of these sorts of tests for AGI is predicated on the hypothesis that the degree of success of an AI program at carrying out some domain-specific task, is correlated with the potential of that program for being developed into a robust AGI program with broad intelligence. If humanlike AGI and problem-area-specific "narrow AI" are in fact very different sorts of pursuits requiring very different principles, as we suspect, then these tests are not strongly relevant to the AGI problem.

There are also some standard evaluation paradigms aimed at AI going beyond specific tasks. For instance, there is a literature on "multitask learning" and "transfer learning," where the goal for an AI is to learn one task quicker given another task solved previously [**?**, TM95, **?**, TS07, RZDK05]. This is one of the capabilities an AI agent will need to simultaneously learn different types of tasks as proposed in the Preschool scenario given here. And there is a literature on "shaping," where the idea is to build up the capability of an AI by training it on progressively more difficult versions of the same tasks [LD03]. Again, this is one sort of capability an AI will need to possess if it is to move up some type of curriculum, such as a school curriculum.

While we applaud the work done on multitask learning and shaping, we feel that exploring these processes using mathematical abstractions, or in the domain of various narrowly-proscribed machine-learning or robotics test problems, may not adequately address the problem of AGI. The problem is that generalization among tasks, or from simpler to more difficult versions of the same task, is a process whose nature may depend strongly on the overall nature of the set of tasks and task-versions involved. Real-world tasks have a subtlety of interconnectedness and developmental course that is not captured in current mathematical learning frameworks nor standard AI test problems.

To put it mathematically, we suggest that the universe of real-world human tasks has a host of "special statistical properties" that have implications regarding what sorts of AI programs will be most suitable; and that, while exploring and formalizing the nature of these statistical properties is important, an easier and more reliable approach to AGI testing is to create a testing environment that embodies these properties implicitly, via its being an emu-

lation of the cognitively meaningful aspects of the real-world human learning environment.

One way to see this point vividly is to contrast the current proposal with the "General Game Player" AI competition, in which AIs seek to learn to play games based on formal descriptions of the rules.[1]. Clearly doing GGP well requires powerful AGI; and doing GGP even mediocrely probably requires robust multitask learning and shaping. But we suspect GGP is far inferior to AGI Preschool as an approach to testing early-stage AI programs aimed at roughly humanlike intelligence. This is because, unlike the tasks involved in AI Preschool, the tasks involved in doing simple instances of GGP seem to have little relationship to humanlike intelligence or real-world human tasks.

16.2 Elements of Preschool Design

What we mean by an "AGI Preschool" is simply a porting to the AGI domain of the essential aspects of human preschools. While there is significant variance among preschools there are also strong commonalities, grounded in educational theory and experience. We will briefly discuss both the physical design and educational curriculum of the typical human preschool, and which aspects transfer effectively to the AGI context.

On the physical side, the key notion in modern preschool design is the "learning center," an area designed and outfitted with appropriate materials for teaching a specific skill. Learning centers are designed to encourage learning by doing, which greatly facilitates learning processes based on reinforcement, imitation and correction (see Chapter 31 of Part 2 for a detailed discussion of the value of this combination); and also to provide multiple techniques for teaching the same skills, to accommodate different learning styles and prevent over-fitting and overspecialization in the learning of new skills.

Centers are also designed to cross-develop related skills. A "manipulatives center," for example, provides physical objects such as drawing implements, toys and puzzles, to facilitate development of motor manipulation, visual discrimination, and (through sequencing and classification games) basic logical reasoning. A "dramatics center," on the other hand, cross-trains interpersonal and empathetic skills along with bodily-kinesthetic, linguistic, and musical skills. Other centers, such as art, reading, writing, science and math centers are also designed to train not just one area, but to center around a primary intelligence type while also cross-developing related areas. For specific examples of the learning centers associated with particular contemporary preschools, see [**?**].

[1] http://games.stanford.edu/

In many progressive, student-centered preschools, students are left largely to their own devices to move from one center to another throughout the preschool room. Generally, each center will be staffed by an instructor at some points in the day but not others, providing a variety of learning experiences. At some preschools students will be strongly encouraged to distribute their time relatively evenly among the different learning centers, or to focus on those learning centers corresponding to their particular strengths and/or weaknesses.

To imitate the general character of a human preschool, one would create several centers in a robot lab or virtual world. The precise architecture will best be adapted via experience but initial centers would likely be:

- **a blocks center**: a table with blocks on it
- **a language center**: a circle of chairs, intended for people to sit around and talk with the robot
- **a manipulatives center**, with a variety of different objects of different shapes and sizes, intended to teach visual and motor skills
- **a ball play center**: where balls are kept in chests and there is space for the robot to kick the balls around
- **a dramatics center** where the robot can observe and enact various movements

16.3 Elements of Preschool Curriculum

While preschool curricula vary considerably based on educational philosophy and regional and cultural factors, there is a great deal of common, shared wisdom regarding the most useful topics and methods for preschool teaching. Guided experiential learning in diverse environments and using varied materials is generally agreed upon as being an optimal methodology to reach a wide variety of learning types and capabilities. Hands-on learning provides grounding in specifics, where as a diversity of approaches allows for generalization.

Core knowledge domains are also relatively consistent, even across various philosophies and regions. Language, movement and coordination, autonomous judgment, social skills, work habits, temporal orientation, spatial orientation, mathematics, science, music, visual arts, and dramatics are universal areas of learning which all early childhood learning touches upon. The particulars of these skills may vary, but all human children are taught to function in these domains. The level of competency developed may vary, but general domain knowledge is provided. For example, most kids won't be the next Maria Callas, Ravi Shankar or Gene Ween, but nearly all learn to hear, understand and appreciate music.

Tables 16.1 - 16.3 review the key capabilities taught in preschools, and identify the most important specific skills that need to be evaluated in the

context of each capability. This table was assembled via surveying the curricula from a number of currently existing preschools employing different methodologies both based on formal academic cognitive theories [?] and more pragmatic approaches, such as: Montessori [?], Waldorf [?], Brain Gym (www.braingym.org) and Core Knowledge (www.coreknowledge.org).

Type of Capability	*Specific Skills to be Evaluated*
Story Understanding	• Understanding narrative sequence • Understanding character development • Dramatize a story • Predict what comes next in a story
Linguistic	• Give simple descriptions of events • Describe similarities and differences • Describe objects and their functions
Linguistic / Spatial-Visual	Interpreting pictures
Linguistic / Social	• Asking questions appropriately • Answering questions appropriately • Talk about own discoveries • Initiate conversations • Settle disagreements • Verbally express empathy • Ask for help • Follow directions
Linguistic / Scientific	• Provide possible explanations for events or phenomena • Carefully describe observations • Draw conclusions from observations

Table 16.1 Categories of Preschool Curriculum, Part 1

16.3.1 Preschool in the Light of Intelligence Theory

Comparing Table 16.1 to Gardner's Multiple Intelligences (MI) framework briefly reviewed in Chapter 2, the high degree of harmony is obvious, and is borne out by more detailed analysis. Preschool curriculum as standardly practiced is very well attuned to MI, and naturally covers all the bases that Gardner identifies as important. And this is not at all surprising since one of

Type of Capability	*Specific Skills to be Evaluated*
Logical-Mathematical	• Categorizing • Sorting • Arithmetic • Performing simple "proto-scientific experiments"
Nonverbal Communication	• Communicating via gesture • Dramatizing situations • Dramatizing needs, wants • Express empathy
Spatial-Visual	• Visual patterning • Self-expression through drawing • Navigate
Objective	• Assembling objects • Disassembling objects • Measurement • Symmetry • Similarity between structures (e.g. block structures and real ones)

Table 16.2 Categories of Preschool Curriculum, Part 2

Type of Capability	*Specific Skills to be Evaluated*
Interpersonal	• Cooperation • Display appropriate behavior in various settings • Clean up belongings • Share supplies
Emotional	• Delay gratification • Control emotional reactions • Complete projects

Table 16.3 Categories of Preschool Curriculum, Part 3

Gardner's key motivations in articulating MI theory was the pragmatics of educating humans with diverse strengths and weaknesses.

Regarding intelligence as "the ability to achieve complex goals in complex environments," it is apparent that preschools are specifically designed to pack a large variety of different micro-environments (the learning centers) into a single room, and to present a variety of different tasks in each environment.

The environments constituted by preschool learning centers are designed as microcosms of the most important aspects of the environments faced by humans in their everyday lives.

16.4 Task-Based Assessment in AGI Preschool

Professional pedagogues such as [**?**] discuss evaluation of early childhood learning as intended to assess both specific curriculum content knowledge as well as the child's learning process. It should be as unobtrusive as possible, so that it just seems like another engaging activity, and the results used to tailor the teaching regimen to use different techniques to address weaknesses and reinforce strengths.

For example, with group building of a model car, students are tested on a variety of skills: procedural understanding, visual acuity, motor acuity, creative problem solving, interpersonal communications, empathy, patience, manners, and so on. With this kind of complex, yet engaging, activity as a metric the teacher can see how each student approaches the process of understanding each subtask, and subsequently guide each student's focus differently depending on strengths and weaknesses.

In Tables 16.4 and 16.5 we describe some particular tasks that AGIs may be meaningfully assigned in the context of a general AGI Preschool design and curriculum as described above. Of course, this is a very partial list, and is intended as evocative rather than comprehensive.

Any one of these tasks can be turned into a rigorous quantitative test, thus allowing the precise comparison of different AGI systems' capabilities; but we have chosen not to emphasize this point here, partly for space reasons and partly for philosophical ones. In some contexts the quantitative comparison of different systems may be the right thing to do, but as discussed in Chapter 17 there are also risks associated with this approach, including the emergence of an overly metrics-focused "bakeoff mentality" among system developers, and overfitting of AI abilities to test taking. What is most important is the isolation of specific tasks on which different systems may be experientially trained and then qualitatively assessed and compared, rather than the evaluation of quantitative metrics.

Task-oriented testing allows for feedback on applications of general pedagogical principles to real-world, embodied activities. This allows for iterative refinement based learning (shaping), and cross development of knowledge acquisition and application (multitask learning). It also helps militate against both cheating, and over-fitting, as teachers can make ad-hoc modifications to the tests to determine if this is happening and correct for it if necessary.

E.g., consider a linguistic task in which the AGI is required to formulate a set of instructions encapsulating a given behavior (which may include components that are physical, social, linguistic, etc.). Note that although this

Intelligence Type	Test
Linguistic	• write a set of instructions • speak on a subject • edit a written piece or work • write a speech • commentate on an event • apply positive or negative 'spin' to astory
Logical-Mathematical	• perform arithmetic calculations • create a process to measure something • analyse how a machine works • create a process • devise a strategy to achieve an aim • assess the value of a proposition
Musical	• perform a musical piece • sing a song • review a musical work • coach someone to play a musical instrument
Bodily-Kinesthetic	• juggle • demonstrate a sports technique • flip a beer-mat • create a mime to explain something • toss a pancake • fly a kite

Table 16.4 Prototypical preschool intelligence assessment tasks, Part 1

is presented as centrally a linguistic task, it actually involves a diverse set of competencies since the behavior to be described may encompass multiple real-world aspects.

To turn this task into a more thorough test one might involve a number of human teachers and a number of human students. Before the test, an ensemble of copies of the AGI would be created, with identical knowledge state. Each copy would interact with a different human teacher, who would demonstrate to it a certain behavior. After testing the AGI on its own knowledge of the material, the teacher would then inform the AGI that it will then be tested on its ability to verbally describe this behavior to another. Then, the teacher goes away and the copy interacts with a series of students, attempting to convey to the students the instructions given by the teacher.

The teacher can thereby assess both the AGI's understanding of the material, and the ability to explain it to the other students. This separates out

Intelligence Type	Test
Spatial-Visual	• design a costume • interpret a painting • create a room layout • create a corporate logo • design a building • pack a suitcase or the trunk of a car
Interpersonal	• interpret moods from facial expressions • demonstrate feelings through body language • affect the feelings of others in a planned way • coach or counsel another

Table 16.5 Prototypical preschool intelligence assessment tasks, Part 2

assessment of understanding from assessment of ability to communicate understanding, attempting to avoid conflation of one with the other. The design of the training and testing needs to account for potential

This testing protocol abstracts away from the particularities of any one teacher or student, and focuses on effectiveness of communication in a human context rather than according to formalized criteria. This is very much in the spirit of how assessment takes place in human preschools (with the exception of the copying aspect): formal exams are rarely given in preschool, but pragmatic, socially-embedded assessments are regularly made.

By including the copying aspect, more rigorous statistical assessments can be made regarding efficacy of different approaches for a given AGI design, independent of past teaching experiences. The multiple copies may, depending on the AGI system design, then be able to be reintegrated, and further "learning" be done by higher-order cognitive systems in the AGI that integrate the disparate experiences of the multiple copies.

This kind of parallel learning is different from both sequential learning that humans do, and parallel presences of a single copy of an AGI (such as in multiple chat rooms type experiments). All three approaches are worthy of study, to determine under what circumstances, and with which AGI designs, one is more successful than another.

It is also worth observing how this test could be tweaked to yield a test of generalization ability. After passing the above, the AGI could then be given a description of a new task (acquisition), and asked to explain the new one (variation). And, part of the training behavior might be carried out unobserved by the AGI, thus requiring the AGI to infer the omitted parts of the task it needs to describe.

Another popular form of early childhood testing is puzzle block games. These kinds of games can be used to assess a variety of important cognitive

skills, and to do so in a fun way that not only examines but also encourages creativity and flexible thinking. Types of games include pattern matching games in which students replicate patterns described visually or verbally, pattern creation games in which students create new patterns guided by visually or verbally described principles, creative interpretation of patterns in which students find meaning in the forms, and free-form creation. Such games may be individual or cooperative.

Cross training and assessment of a variety of skills occurs with pattern block games: for example, interpretation of visual or linguistic instructions, logical procedure and pattern following, categorizing, sorting, general problem solving, creative interpretation, experimentation, and kinematic acuity. By making the games cooperative, various interpersonal skills involving communication and cooperation are also added to the mix.

The puzzle block context bring up some general observations about the role of kinematic and visuospatial intelligence in the AGI Preschool. Outside of robotics and computer vision, AI research has often downplayed these sorts of intelligence (though, admittedly, this is changing in recent years, e.g. with increasing research focus on diagrammatic reasoning). But these abilities are not only necessary to navigate real (or virtual) spatial environments. They are also important components of a coherent, conceptually well-formed understanding of the world in which the student is embodied. Integrative training and assessment of both rigorous cognitive abilities generally most associated with both AI and "proper schooling" (such as linguistic and logical skills) along with kinematic and aesthetic/sensory abilities is essential to the development of an intelligence that can successfully both operate in and sensibly communicate about the real world in a roughly humanlike manner. Whether or not an AGI is targeted to interpret physical-world spatial data and perform tasks via robotics, in order to communicate ideas about a vast array of topics of interest to any intelligence in this world, an AGI must develop aspects of intelligence other than logical and linguistic cognition.

16.5 Beyond Preschool

Once an AGI passes preschool, what are the next steps? There is still a long way to go, from preschool to an AGI system that is capable of, say, passing the Turing Test or serving as an effective artificial scientist.

Our suggestion is to extend the school metaphor further, and make use of existing curricula for higher levels of virtual education: grade school, secondary school, and all levels of post-secondary education. If an AGI can pass online primary and secondary schools such as e-tutor.com, and go on to earn an online degree from an accredited university, then clearly said AGI has successfully achieved "human level, roughly humanlike AGI." This sort of testing is interesting not only because it allows assessment of stages intermediate be-

tween preschool and adult, but also because it tests humanlike intelligence without requiring precise imitation of human behavior.

If an AI can get a BA degree at an accredited university, via online coursework (assuming for simplicity courses where no voice interaction is needed), then we should consider that AI to have human-level intelligence. University coursework spans multiple disciplines, and the details of the homework assignments and exams are not known in advance, so like a human student the AGI team can't cheat.

In addition to the core coursework, a schooling approach also tests basic social interaction and natural language communication, ability to do online research, and general problem solving ability. However, there is no rigid requirement to be strictly humanlike in order to pass university classes.

Most of our concrete examples in the following chapters will pertain to the preschool context, because it's simple to understand, and because we feel that getting to the "AGI preschool student" level is going to be the largest leap. Once that level is obtained, moving further will likely be difficult also, but we suspect it will be more a matter of steady incremental improvements – whereas the achievement of preschool-level functionality will be a large leap from the current situation.

16.6 Issues with Virtual Preschool Engineering

As noted above there are two broad approaches to realizing the "AGI Preschool" idea: using the AGI to control a physical robot and then crafting a preschool environment suitable to the robot's sensors and actuators; or, using the AGI to control a virtual agent in an appropriately rich virtual-world preschool. The robotic approach is harder from an AI perspective (as one must deal with problems of sensation and actuation), but easier from an environment-construction perspective. In the virtual world case, one quickly runs up against the current limitations of virtual world technologies, which have been designed mainly for entertainment or social-networking purposes, not with the requirements of AGI systems in mind.

In Chapter 9 we discussed the general requirements that an environment should possess to be supportive of humanlike intelligence. Referring back to that list, it's clear that current virtual worlds are fairly strong on multimodal communication, and fairly weak on naive physics. More concretely, if one wants a virtual world so that

1. one could carry out all the standard cognitive development experiments described in developmental psychology books
2. one could implement intuitively reasonable versions of all the standard activities in all the standard learning stations in a contemporary preschool

then current virtual world technologies appear not to suffice.

As reviewed above, typical preschool activities include for instance building with blocks, playing with clay, looking in a group at a picture book and hearing it read aloud, mixing ingredients together, rolling/throwing/catching balls, playing games like tag, hide-and-seek, Simon Says or Follow the Leader, measuring objects, cutting paper into different shapes, drawing and coloring, etc.

And, as typical, not necessarily representative examples of tasks psychologists use to measure cognitive development (drawn mainly from the Piagetan tradition, without implying any assertion that this is the only tradition worth pursuing), consider the following:

1. Which row has more circles- A or B? A: O O O O O, B: OOOOO
2. If Mike is taller than Jim, and Jim is shorter than Dan, then who is the shortest? Who is the tallest?
3. Which is heavier- a pound of feathers or a pound of rocks?
4. Eight ounces of water is poured into a glass that looks like the fat glass in Figure 2 16.1 and then the same amount is poured into a glass that looks like the tall glass in Figure 16.2 . Which glass has more water?
5. A lump of clay is rolled into a snake. All the clay is used to make the snake. Which has more clay in it – the lump or the snake?
6. There are two dolls in a room, Sally and Ann, each of which has her own box, with a marble hidden inside. Sally goes out for a minute, leaving her box behind; and Ann decides to play a trick on Sally: she opens Sally's box, removes the marble, hiding it in her own box. Sally returns, unaware of what happened. Where will Sally would look for her marble?
7. Consider this rule about a set of cards that have letters on one side and numbers on the other: "If a card has a vowel on one side, then it has an even number on the other side." If you have 4 cards labeled "E K 4 7", which cards do you need to turn over to tell if this rule is actually true?
8. Design an experiment to figure out how to make a pendulum that swings more slowly versus less slowly

What we see from this ad hoc, partial list is that a lot of naive physics *is* required to make an even vaguely realistic preschool. A lot of preschool education is about the intersection between abstract cognition and naive physics. A more careful review of the various tasks involved in preschool education bears out this conclusion.

With this in mind, in this section we will briefly describe an approach to extending current virtual world technologies that appears to allow the construction of a reasonably rich and realistic AGI preschool environment, without requiring anywhere near a complete simulation of realistic physics.

16.6.1 Integrating Virtual Worlds with Robot Simulators

One glaring deficit in current virtual world platforms is the lack of flexibility in terms of tool use. In most of these systems today, an avatar can pick up or utilize an object, or two objects can interact, only in specific, pre-programmed ways. For instance, an avatar might be able to pick up a virtual screwdriver only by the handle, rather than by pinching the blade betwen its fingers. This places severe limits on creative use of tools, which is absolutely critical in a preschool context. The solution to this problem is clear: adapt existing generalized physics engines to mediate avatar-object and object-object interactions. This would require more computation than current approaches, but not more than is feasible in a research context.

One way to achieve this goal would be to integrate a robot simulator with a virtual world or game engine, for instance to modify the OpenSim (opensimulator.org) virtual world to use the Gazebo (playerstage.sourceforge.net) robot simulator in place of its current physics engine. While tractable, such a project would require considerable software engineering effort.

16.6.2 BlocksNBeads World

Another glaring deficit in current virtual world platforms is their inability to model physical phenomena besides rigid objects with any sophistication. In this section we propose a potential solution to this issue: a novel class of virtual worlds called BlocksNBeadsWorld, consisting of the following aspects:

1. 3D blocks of various shapes and sizes and frictional coefficients, that can be stacked
2. Adhesive that can be used to stick blocks together, and that comes in two types, one of which can be removed by an adhesive-removing substance, one of which cannot (though its bonds can be broken via sufficient application of force)
3. Spherical beads, each of which has intrinsic unchangeable adhesion properties defined according to a particular, simple "adhesion logic"
4. Each block, and each bead, may be associated with multidimensional quantities representing its taste and smell; and may be associated with a set of sounds that are made when it is impacted with various forces at various positions on its surface

Interaction between blocks and beads is to be calculated according to standard Newtonian physics, which would be compute-intensive in the case of a large number of beads, but tractable using distributed processing. For instance if 10K beads were used to cover a humanoid agent's face, this would provide a fairly wide diversity of facial expressions; and if 10K beads were

used to form a blanket laid on a bed, this would provide a significant amount of flexibility in terms of rippling, folding and so forth. Yet, this order of magnitude of interactions is very small compared to what is done in contemporary simulations of fluid dynamics or, say, quantum chromodynamics.

One key aspect of the spherical beads is that they can be used to create a variety of rigid or flexible surfaces, which may exist on their own or be attached to blocks-based constructs. The specific inter-bead adhesion properties of the beads could be defined in various ways, and will surely need to be refined via experimentation, but a simple scheme that seems to make sense is as follows.

Each bead can have its surface tesselated into hexagons (the number of these can be tuned), and within each hexagon it can have two different adhesion coefficients: one for adhesion to other beads, and one for adhesion to blocks. The adhesion between two beads along a certain hexagon is then determined by their two adhesion coefficients; and the adhesion between a bead and a block is determined by the adhesion coefficient of the bead, and the adhesion coefficient of the adhesive applied to the block. A distinction must be drawn between rigid and flexible adhesion: rigid adhesion sticks a bead to something in a way that can't be removed except via breaking it off; whereas flexible adhesion just keeps a bead very close to the thing it's stuck onto. Any two entities may be stuck together either rigidly or flexibly. Sets of beads with flexible adhesion to each other can be used to make entities like strings, blankets or clothes.

Using the above adhesion logic, it seems one could build a wide variety of flexible structures using beads, such as (to give a very partial list):

1. fabrics with various textures, that can be draped over blocks structures,
2. multilayered coatings to be attached to blocks structures, serving (among many other examples) as facial expressions
3. liquid-type substances with varying viscosities, that can be poured between different containers, spilled, spread, etc.
4. strings tyable in knots; rubber bands that can be stretched; etc.

Of course there are various additional features one could add. For instance one could add a special set of rules for vibrating strings, allowing BlocksNBeadsWorld to incorporate the creation of primitive musical instruments. Variations like this could be helpful but aren't necessary for the world to serve its essential purpose.

Note that one does not have true fluid dynamics in BlocksNBeadsWorld, but, it seems that the latter is not necessary to encompass the phenomena covered in cognitive developmental tests or preschool tasks. The tests and tasks that are done with fluids can instead be done with masses of beads. For example, consider the conservation of volume task shown in Figures 16.1 and 16.2 below: it's easy enough to envision this being done with beads rather than milk. Even a few hundred beads is enough to be psychologically perceived as a mass rather than a set of discrete units, and to be manipulated

and analyzed as such. And the simplification of not requiring fluid mechanics in one's virtual world is immense.

Next, one can implement equations via which the adhesion coefficients of a bead are determined in part by the adhesion coefficients of nearby beads, or beads that are nearby in certain directions (with direction calculated in local spherical coordinates). This will allow for complex cracking and bending behaviors – not identical to those in the real world, but with similar qualitative characteristics. For example, without this feature one could create paperlike substances that could be cut with scissors – but *with* this feature, one could go further and create woodlike substances that would crack when nails were hammered into them in certain ways, and so forth.

Further refinements are certainly possible also. One could add multidimensional adhesion coefficients, allowing more complex sorts of substances. One could allow beads to vibrate at various frequencies, which would lead to all sorts of complex wave patterns in bead compounds. Etc. In each case, the question to be asked is: what important cognitive abilities are dramatically more easily learnable in the presence of the new feature than in its absence?

The combination of blocks and beads seems ideal for implementing a more flexible and AGI-friendly type of virtual body than is currently used in games and virtual worlds. One can easily envision implementing a body with

1. a skeleton whose bones consist of appropriately shaped blocks
2. joints consisting of beads, flexibly adhered to the bones
3. flesh consisting of beads, flexibly adhered to each other
4. internal "plumbing" consisting of tubes whose walls are beads rigidly adhered to each other, and flexibly adhered to the surrounding flesh (the plumbing could then serve to pass beads through, where slow passage would be ensured by weak adhesion between the walls of the tubes and the beads passing through the tubes)

This sort of body would support rich kinesthesia; and rich, broad analogy-drawing between the internally-experienced body and the externally-experienced world. It would also afford many interesting opportunities for flexible movement control. Virtual animals could be created along with virtual humanoids.

Regarding the extended mind, it seems clear that blocks and beads are adequate for the creation of a variety of different tools. Equipping agents with "glue guns" able to affect the adhesive properties of both blocks and beads would allow a diversity of building activity; and building with masses of beads could become a highly creative activity. Furthermore, beads with appropriately specified adhesion (within the framework outlined above) could be used to form organically growing plant-like substances, based on the general principles used in L-system models of plant growth (Prusinciewicz and Lindenmayer 1991). Structures with only beads would vaguely resemble herbaceous plants; and structures involving both blocks and beads would more resemble woody plants. One could even make organic structures that flourish or

otherwise based on the light available to them (without of course trying to simulate the chemistry of photosynthesis).

Some elements of chemistry may be achieved as well, though nowhere near what exists in physical reality. For instance, melting and boiling at least should be doable: assign every bead a temperature, and let solid interbead bonds turn liquid above a certain temperature and disappear completely above some higher temperature. You could even have a simple form of fire. Let fire be an element, whose beads have negative gravitational mass. Beads of fuel elements like wood have a threshold temperature above which they will turn into fire beads, with release of additional heat.[2]

The philosophy underlying these suggested bead dynamics is somewhat comparable to that outlined in Wolfram's book *A New Kind of Science* [Wol02]. There he proposes cellular automata models that emulate the qualitative characteristics of various real-world phenomena, without trying to match real-world data precisely. For instance, some of his cellular automata demonstrate phenomena very similar to turbulent fluid flow, without implementing the Navier-Stokes equations of fluid dynamics or trying to precisely match data from real-world turbulence. Similarly, the beads in BlocksNBeadsWorld are intended to qualitatively demonstrate the real-world phenomena most useful for the development of humanlike embodied intelligence, without trying to precisely emulate the real-world versions of these phenomena.

The above description has been left imprecisely specified on purpose. It would be straightforward to write down a set of equations for the block and bead interactions, but there seems little value in articulating such equations without also writing a simulation involving them and testing the ensuing properties. Due to the complex dynamics of bead interactions, the fine-tuning of the bead physics is likely to involve some tuning based on experimentation, so that any equations written down now would likely be revised based on experimentation anyway. Our goal here has been to outline a certain class of potentially useful environments, rather than to articulate a specific member of this class.

Without the beads, BlocksNBeadsWorld would appear purely as a "Blocks World with Glue" – essentially a substantially upgraded version of the Blocks Worlds frequently used in AI, since first introduced in [Win72]. Certainly a pure "Blocks World with Glue" would have greater simplicity than BlocksNBeadsWorld, and greater richness than standard Blocks World; but this simplicity comes with too many limitations, as shown by consideration of the various naive physics requirements inventoried above. One simply cannot run the full spectrum of humanlike cognitive development experiments, or preschool educational tasks, using blocks and glue alone. One can try to create analogous tasks using only blocks and glue, but this quickly becomes

[2] Thanks are due to Russell Wallace for the suggestions in this paragraph

extremely awkward. Whereas in the BlocksNBeadsWorld the capability for this full spectrum of experiments and tasks seems to fall out quite naturally.

What's missing from BlocksNBeadsWorld should be fairly obvious. There isn't really any distinction between a fluid and a powder: there are masses, but the types and properties of the masses are not the same as in the real world, and will surely lack the nuances of real-world fluid dynamics. Chemistry is also missing: processes like cooking and burning, although they can be crudely emulated, will not have the same richness as in the real world. The full complexity of body processes is not there: the body-design method mentioned above is far richer and more adaptive and responsive than current methods of designing virtual bodies in 3DSMax or Maya and importing them into virtual world or game engines, but still drastically simplistic compared to real bodies with their complex chemical signaling systems and couplings with other bodies and the environment. The hypothesis we're making in this section is that these lacunae aren't that important from the point of view of humanlike cognitive development. We suggest that the key features of naive physics and folk psychology enumerated above can be mastered by an AGI in BlocksNBeadsWorld in spite of its limitations, and that – together with an appropriate AGI design – this probably suffices for creating an AGI with the inductive biases constituting humanlike intelligence.

To drive this point home more thoroughly, consider three potential virtual world scenarios:

1. A world containing realistic fluid dynamics, where a child can pour water back and forth between two cups of different shapes and sizes, to understand issues such as conservation of volume
2. A world more like today's Second Life, where fluids don't really exist, and things like lakes are simulated via very simple rules, and pouring stuff back and forth between cups doesn't happen unless it's programmed into the cups in a very specialized way
3. A BlocksNBeadsWorld type world, where a child can pour masses of beads back and forth between cups, but not masses of liquid

Our qualitative judgment is that Scenario 3 is going to allow a young AI to gain the same essential insights as Scenario 1, whereas Scenario 2 is just too impoverished. I have explored dozens of similar scenarios regarding different preschool tasks or cognitive development experiments, and come to similar conclusions across the board. Thus, our current view is that something like BlocksNBeadsWorld can serve as an adequate infrastructure for an AGI Preschool, supporting the development of human-level, roughly human-like AGI.

And, if this view turns out to be incorrect, and BlocksNBeadsWorld is revealed as inadequate, then we will very likely still advocate the conceptual approach enunciated above as a guide for designing virtual worlds for AGI. That is, we would suggest to explore the hypothetical failure of BlocksNBeadsWorld via asking two questions:

1. Are there basic naive physics or folk psychology requirements that were missed in creating the specifications, based on which the adequacy of BlocksNBeadsWorld was assessed?
2. Does BlocksNBeadsWorld fail to sufficiently emulate the real world in respect to some of the articulated naive physics or folk psychology requirements?

The answers to these questions would guide the improvement of the world or the design of a better one.

Regarding the practical implementation of BlocksNBeadsWorld, it seems clear that this is within the scope of modern game engine technology, however, it is not something that could be encompassed within an existing game or world engine without significant additions; it would require substantial customengineering. There exist commodity and open-source physics engines that efficiently carry out Newtonian mechanics calculations; while they might require some tuning and extension to handle BlocksNBeadWorld, the main issue would be achieving adequate speed of physics calculation, which given current technology would need to be done via modifying existing engines to appropriately distribute processing among multiple GPUs.

Finally, an additional avenue that merits mention is the use of BlocksNBeads physics internally within an AGI system, as part of an internal simulation world that allows it to make "mind's eye" estimative simulations of real or hypothetical physical situations. There seems no reason that the same physics software libraries couldn't be used both for the external virtual world that the AGI's body lives in, and for an internal simulation world that the AGI uses as a cognitive tool. In fact, the BlocksNBeads library could be used as an internal cognitive tool by AGI systems controlling physical robots as well. This might require more tuning of the bead dynamics to accord with the dynamics of various real-world systems; but, this tuning would be beneficial for the BlocksNBeadWorld as well.

Fig. 16.1 Part 1 of a Piagetan conservation of volume experiment: a child observes that two glasses obviously have the same amount of milk in them, and then sees the content of one of the glasses poured into a different-shaped glass.

Fig. 16.2 Part 2 of a Piagetan conservation of volume experiment: a child observes two different-shaped glasses, which (depending on the level of his cognition), he may be able to infer have the same amount of milk in them, due to the events depicted in Figure 16.1.

Chapter 17
A Preschool-Based Roadmap to Advanced AGI

17.1 Introduction

Supposing the CogPrime approach to creating advanced AGI is workable – then what are the right practical steps to follow? The various structures and algorithms outlined in Part 2 of this book should be engineered and software-tested, of course – but that's only part of the study. The AGI system implemented will need to be taught, and it will need to be placed in situations where it can develop an appropriate self-model and other critical internal network structures. The complex structures and algorithms involved will need to be fine-tuned in various ways, based on qualitatively observing the overall system's behavior in various situations. To get all this right without excessive confusion or time-wastage requires a fairly clear *roadmap* for CogPrime development.

In this chapter we'll sketch one particular roadmap for the development of human-level, roughly human-like AGI – which we're not selling as the only one, or even necessarily as the best one. It's just one roadmap that we have thought about a lot, and that we believe has a strong chance of proving effective. Given resources to pursue only one path for AGI development and teaching, this would be our choice, at present. The roadmap outlined here is not restricted to CogPrime in any highly particular ways, but it has been developed largely with CogPrime in mind; those developing other AGI designs could probably use this roadmap just fine, but might end up wanting to make various adjustments based on the strengths and weaknesses of their own approach.

What we mean here by a "roadmap" is, in brief: a sequence of "milestone" tasks, occurring in a small set of common environments or "scenarios," organized so as to lead to a commonly agreed upon set of long-term goals. I.e., what we are after here is a "capability roadmap" – a roadmap laying out a series of capabilities whose achievement seems likely to lead to human-level

AGI. Other sorts of roadmaps such as "tools roadmaps" may also be valuable, but are not our concern here.

More precisely, we confront the task of roadmapping by identifying scenarios in which to embed our AGI system, and then "competency areas" in which the AGI system must be evaluated. Then, we envision a roadmap as consisting of a set of one or more task-sets, where each task set is formed from a combination of a scenario with a list of competency areas. To create a task-set one must choose a particular scenario, and then articulate a set of specific tasks, each one addressing one or more of the competency areas. Each task must then get associated with particular performance metrics – quantitative wherever possible, but perhaps qualitative in some cases depending on the nature of the task. Here we give a partial task-set for the "virtual and robot preschool" scenarios discussed in Chapter 16, and a couple example quantitative metrics just to illustrate what is intended; the creation of a fully detailed roadmap based on the ideas outlined here is left for future work

The train of thought presented in this chapter emerged in part from a series of conversations preceding and during the "AGI Roadmap Workshop" held at the University of Tennessee, Knoxville in October 2008. Some of the ideas also trace back to discussions held during two workshops on "Evaluation and Metrics for Human-Level AI " organized by John Laird and Pat Langley (one in Ann Arbor in late 2008, and one in Tempe in early 2009). Some of the conclusions of the Ann Arbor workshop were recorded in [LWML09]. Inspiration was also obtained from discussion at the "Future of AGI" post-conference workshop of the AGI-09 conference, triggered by Itamar ArelÕs [ARK09a] presentation on the "AGI Roadmap" theme; and from an earlier article on AGI Roadmapping by [**?**].

However, the focus of the AGI Roadmap Workshop was considerably more general than the present chapter. Here we focus on preschool-type scenarios, whereas at the workshop a number of scenarios were discussed, including the preschool scenarios but also, for example,

- Standardized Tests and School Curricula
- Elementary, Middle and High School Student
- General Videogame Learning
- Wozniak's Coffee Test: go into a random American house and figure out how to make coffee, and do it
- Robot College Student
- General Call Center Respondent

For each of these scenarios, one may generate tasks corresponding to each of the competency areas we will outline below. CogPrime is applicable in all these scenarios, so our choice to focus on preschool scenarios is an additional judgment call beyond those judgment calls required to specify the CogPrime design. The roadmap presented here is a "AGI Preschool Roadmap" and as such is a special case of the broader "AGI Roadmap" outlined at the workshop.

17.2 Measuring Incremental Progress Toward Human-Level AGI

In Chapter 2, we discussed several examples of practical goals that we find to plausibly characterize "human level AGI", e.g.

- *Turing Test*
- *Virtual World Turing Test*
- *Online University Test*
- *Physical University Test*
- *Artificial Scientist Test*

We also discussed our optimism regarding possibility that in the future AGI may advance beyond the human level, rendering all these goals "early-stage subgoals."

However, in this chapter we will focus our attention on the nearer term. The above goals are ambitious ones, and while one can talk a lot about how to precisely measure their achievement, we don't feel that's the most interesting issue to ponder at present. More critical is to think about how to measure incremental progress. How do you tell when you're 25% or 50% of the way to having an AGI that can pass the Turing Test, or get an online university degree. Fooling 50% of the Turing Test judges is not a good measure of being 50% of the way to passing the Turing Test (that's too easy); and passing 50% of university classes is not a good measure of being 50% of the way to getting an online university degree (it's too hard – if one had an AGI capable of doing that, one would almost surely be very close to achieving the end goal). Measuring incremental progress toward human-level AGI is a subtle thing, and we argue that the best way to do it is to focus on particular scenarios and the achievement of specific competencies therein.

As we argued in Chapter 8 there are some theoretical reasons to doubt the possibility of creating a rigorous objective test for partial progress toward AGI – a test that would be convincing to skeptics, and impossible to "game" via engineering a system specialized to the test. Fortunately, though we don't need a test of this nature for the purposes of assessing our own incremental progress toward advanced AGI, based on our knowledge about our own approach.

Based on the nature of the grand goals articulated above, there seems to be a very natural approach to creating a set of incremental capabilities building toward AGI: *to draw on our copious knowledge about human cognitive development*. This is by no means the only possible path; one can envision alternatives that have nothing to do with human development (and those might also be better suited to non-human AGIs). However, so much detailed knowledge about human development is available – as well as solid knowledge that the human developmental trajectory does lead to human-level AI – that the motivation to draw on human cognitive development is quite strong.

The main problem with the human development inspired approach is that cognitive developmental psychology is not as systematic as it would need to be for AGI to be able to translate it directly into architectural principles and requirements. As noted above, while early thinkers like Piaget and Vygotsky outlined systematic theories of child cognitive development, these are no longer considered fully accurate, and one currently faces a mass of detailed theories of various aspects of cognitive development, but without an unified understanding. Nevertheless we believe it is viable to work from the human-development data and understanding currently available, and craft a workable AGI roadmap therefrom.

With this in mind, what we give next is a fairly comprehensive list of the competencies that we feel AI systems should be expected to display in one or more of these scenarios in order to be considered as full-fledged "human level AGI" systems. These competency areas have been assembled somewhat opportunistically via a review of the cognitive and developmental psychology literature as well as the scope of the current AI field. We are not claiming this as a precise or exhaustive list of the competencies characterizing human-level general intelligence, and will be happy to accept additions to the list, or mergers of existing list items, etc. What we are advocating is not this specific list, but rather the approach of enumerating competency areas, and then generating tasks by combining competency areas with scenarios.

We also give, with each competency, an example task illustrating the competency. The tasks are expressed in the robot preschool context for concreteness, but they all apply to the virtual preschool as well. Of course, these are only examples, and ideally to teach an AGI in a structured way one would like to

- associate several tasks with each competency
- present each task in a graded way, with multiple subtasks of increasing complexity
- associate a quantitative metric with each task

However, the briefer treatment given here should suffice to give a sense for how the competencies manifest themselves practically in the AGI Preschool context.

1. Perception
    - **Vision**: image and scene analysis and understanding
        - *Example task:* When the teacher points to an object in the preschool, the robot should be able to identify the object and (if it's a multi-part object) its major parts. If it can't perform the identification initially, it can approach the object and manipulate it before making its identification.
    - **Hearing**: identifying the sounds associated with common objects; understanding which sounds come from which sources in a noisy environment

  - *Example task:* When the teacher covers the robot's eyes and then makes a noise with an object, the robot should be able to guess what the object is
- **Touch**: identifying common objects and carrying out common actions using touch alone
  - *Example task:* With its eyes and ears covered, the robot should be able to identify some object by manipulating it; and carry out some simple behaviors (say, putting a block on a table) via touch alone
- **Crossmodal**: Integrating information from various senses
  - *Example task:* Identifying an object in a noisy, dim environment via combining visual and auditory information
- **Proprioception**: Sensing and understanding what its body is doing
  - *Example task:* The teacher moves the robot's body into a certain configuration. The robot is asked to restore its body to an ordinary standing position, and then repeat the configuration that the teacher moved it into.

2. Actuation

- **Physical skills**: manipulating familiar and unfamiliar objects
  - *Example task:* Manipulate blocks based on imitating the teacher: e.g. pile two blocks atop each other, lay three blocks in a row, etc.
- **Tool use**, including the flexible use of ordinary objects as tools
  - *Example task:* Use a stick to poke a ball out of a corner, where the robot cannot directly reach
- **Navigation**, including in complex and dynamic environments
  - *Example task:* Find its own way to a named object or person through a crowded room with people walking in it and objects laying on the floor.

3. Memory

- **Declarative**: noticing, observing and recalling facts about its environment and experience
  - *Example task:* If certain people habitually carry certain objects, the robot should remember this (allowing it to know how to find the objects when the relevant people are present, even much later)
- **Behavioral**: remembering how to carry out actions
  - *Example task:* If the robot is taught some skill (say, to fetch a ball), it should remember this much later
- **Episodic**: remembering significant, potentially useful incidents from life history
  - *Example task:* Ask the robot about events that occurred at times when it got particularly much, or particularly little, reward for its

actions; it should be able to answer simple questions about these, with significantly more accuracy than about events occurring at random times

4. Learning
   - **Imitation**: Spontaneously adopt new behaviors that it sees others carrying out
     - *Example task:* Learn to build towers of blocks by watching people do it
   - **Reinforcement**: Learn new behaviors from positive and/or negative reinforcement signals, delivered by teachers and/or the environment
     - *Example task:* Learn which box the red ball tends to be kept in, by repeatedly trying to find it and noticing where it is, and getting rewarded when it finds it correctly
   - **Imitation/Reinforcement**
     - *Example task:* Learn to play "fetch", "tag" and "follow the leader" by watching people play it, and getting reinforced on correct behavior
   - **Interactive Verbal Instruction**
     - *Example task:* Learn to build a particular structure of blocks faster based on a combination of imitation, reinforcement and verbal instruction, than by imitation and reinforcement without verbal instruction
   - **Written Media**
     - *Example task:* Learn to build a structure of blocks by looking at a series of diagrams showing the structure in various stages of completion
   - **Learning via Experimentation**
     - *Example task:* Ask the robot to slide blocks down a ramp held at different angles. Then ask it to make a block slide fast, and see if it has learned how to hold the ramp to make a block slide fast.
5. Reasoning
   - **Deduction**, from uncertain premises observed in the world
     - *Example task:* If Ben more often picks up red balls than blue balls, and Ben is given a choice of a red block or blue block to pick up, which is he more likely to pick up?
   - **Induction**, from uncertain premises observed in the world
     - *Example task:* If Ben comes into the lab every weekday morning, then is Ben likely to come to the lab today (a weekday) in the morning?
   - **Abduction**, from uncertain premises observed in the world
     - *Example task:* If women more often give the robot food than men, and then someone of unidentified gender gives the robot food, is this person a man or a woman?

- **Causal reasoning**, from uncertain premises observed in the world
  - *Example task:* If the robot knows that knocking down Ben's tower of blocks makes him mad, then what will it say when asked if kicking the ball at Ben's tower of blocks will make Ben mad?
- **Physical reasoning**, based on observed "fuzzy rules" of naive physics
  - *Example task:* Given two balls (one rigid and one compressible) and two tunnels (one significantly wider than the balls, one slightly narrower than the balls), can the robot guess which balls will fit through which tunnels?
- **Associational reasoning**, based on observed spatiotemporal associations
  - *Example task:* If Ruiting is normally seen near Shuo, then if the robot knows where Shuo is, that is where it should look when asked to find Ruiting

6. Planning

- **Tactical**
  - *Example task:* The robot is asked to bring the red ball to the teacher, but the red ball is in the corner where the robot can't reach it without a tool like a stick. The robot knows a stick is in the cabinet so it goes to the cabinet and opens the door and gets the stick, and then uses the stick to get the red ball, and then brings the red ball to the teacher.
- **Strategic**
  - *Example task:* Suppose that Matt comes to the lab infrequently, but when he does come he is very happy to see new objects he hasn't seen before (and suppose the robot likes to see Matt happy). Then when the robot gets a new object Matt has not seen before, it should put it away in a drawer and be sure not to lose it or let anyone take it, so it can show Matt the object the next time Matt arrives.
- **Physical**
  - *Example task:* To pick up a cup with a handle which is lying on its side in a position where the handle can't be grabbed, the robot turns the cup in the right position and then picks up the cup by the handle
- **Social**
  - *Example task:* The robot is given a job of building a tower of blocks by the end of the day, and he knows Ben is the most likely person to help him, and he knows that Ben is more likely to say "yes" to helping him when Ben is alone. He also knows that Ben is less likely to say yes if he's asked too many times, because Ben doesn't like being nagged. So he waits to ask Ben till Ben is alone in the lab.

7. Attention
    - **Visual Attention** within its observations of its environment
        - *Example task:* The robot should be able to look at a scene (a configuration of objects in front of it in the preschool) and identify the key objects in the scene and their relationships.
    - **Social Attention**
        - *Example task:* The robot is having a conversation with Itamar, which is giving the robot reward (for instance, by teaching the robot useful information). Conversations with other individuals in the room have not been so rewarding recently. But Itamar keeps getting distracted during the conversation, by talking to other people, or playing with his cellphone. The robot needs to know to keep paying attention to Itamar even through the distractions.
    - **Behavioral Attention**
        - *Example task:* The robot is trying to navigate to the other side of a crowded room full of dynamic objects, and many interesting things keep happening around the room. The robot needs to largely ignore the interesting things and focus on the movements that are important for its navigation task.
8. Motivation
    - **Subgoal creation**, based on its preprogrammed goals and its reasoning and planning
        - *Example task:* Given the goal of pleasing Hugo, can the robot learn that telling Hugo facts it has learned but not told Hugo before, will tend to make Hugo happy?
    - **Affect-based motivation**
        - *Example task:* Given the goal of gratifying its curiosity, can the robot figure out that when someone it's never seen before has come into the preschool, it should watch them because they are more likely to do something new?
    - **Control of emotions**
        - *Example task:* When the robot is very curious about someone new, but is in the middle of learning something from its teacher (who it wants to please), can it control its curiosity and keep paying attention to the teacher?
9. Emotion
    - **Expressing Emotion**
        - *Example task:* Cassio steals the robot's toy, but Ben gives it back to the robot. The robot should appropriately display anger at Cassio, and gratitude to Ben.
    - **Understanding Emotion**

  - *Example task:* Cassio and the robot are both building towers of blocks. Ben points at Cassio's tower and expresses happiness. The robot should understand that Ben is happy with Cassio's tower.

10. Modeling Self and Other

- **Self-Awareness**
  - *Example task:* When someone asks the robot to perform an act it can't do (say, reaching an object in a very high place), it should say so. When the robot is given the chance to get an equal reward for a task it can complete only occasionally, versus a task it finds easy, it should choose the easier one.
- **Theory of Mind**
  - *Example task:* While Cassio is in the room, Ben puts the red ball in the red box. Then Cassio leaves and Ben moves the red ball to the blue box. Cassio returns and Ben asks him to get the red ball. The robot is asked to go to the place Cassio is about to go.
- **Self-Control**
  - *Example task:* Nasty people come into the lab and knock down the robot's towers, and tell the robot he's a bad boy. The robot needs to set these experiences aside, and not let them impair its self-model significantly; it needs to keep on thinking it's a good robot, and keep building towers (that its teachers will reward it for).
- **Other-Awareness**
  - *Example task:* If Ben asks Cassio to carry out a task that the robot knows Cassio cannot do or does not like to do, the robot should be aware of this, and should bet that Cassio will not do it.
- **Empathy**
  - *Example task:* If Itamar is happy because Ben likes his tower of blocks, or upset because his tower of blocks is knocked down, the robot should express and display these same emotions

11. Social Interaction

- **Appropriate Social Behavior**
  - *Example task:* The robot should learn to clean up and put away its toys when it's done playing with them.
- **Social Communication**
  - *Example task:* The robot should greet new human entrants into the lab, but if it knows the new entrants very well and it's busy, it may eschew the greeting
- **Social Inference** about simple social relationships
  - *Example task:* The robot should infer that Cassio and Ben are friends because they often enter the lab together, and often talk to each other while they are there

- **Group Play** at loosely-organized activities
  - *Example task:* The robot should be able to participate in "informally kicking a ball around" with a few people, or in informally collaboratively building a structure with blocks

12. Communication

- **Gestural communication** to achieve goals and express emotions
  - *Example task:* If the robot is asked where the red ball is, it should be able to show by pointing its hand or finger
- **Verbal communication** using English in its life-context
  - *Example tasks:* Answering simple questions, responding to simple commands, describing its state and observations with simple statements
- **Pictorial Communication** regarding objects and scenes it is familiar with
  - *Example task:* The robot should be able to draw a crude picture of a certain tower of blocks, so that e.g the picture looks different for a very tall tower and a wide low one
- **Language acquisition**
  - *Example task:* The robot should be able to learn new words or names via people uttering the words while pointing at objects exemplifying the words or names
- **Cross-modal communication**
  - *Example task:* If told to "touch Bob's knee" but the robot doesn't know what a knee is, being shown a picture of a person and pointed out the knee in the picture should help it figure out how to touch Bob's knee

13. Quantitative

- **Counting** sets of objects in its environment
  - *Example task:* The robot should be able to count small (homogeneous or heterogeneous) sets of objects
- **Simple, grounded arithmetic** with small numbers
  - *Example task:* Learning simple facts about the sum of integers under 10 via teaching, reinforcement and imitation
- **Comparison** of observed entities regarding quantitative properties
  - *Example task:* Ability to answer questions about which object or person is bigger or taller
- **Measurement** using simple, appropriate tools
  - *Example task:* Use of a yardstick to measure how long something is

14. Building/Creation

- **Physical**: creative constructive play with objects

  - *Example task:* Ability to construct novel, interesting structures from blocks
- **Conceptual** invention: concept formation
  - *Example task:* Given a new category of objects introduced into the lab (e.g. hats, or pets), the robot should create a new internal concept for the new category, and be able to make judgments about these categories (e.g. if Ben particularly likes pets, it should notice this after it has identified "pets" as a category)
- **Verbal** invention
  - *Example task:* Ability to coin a new word or phrase to describe a new object (e.g. the way Alex the parrot coined "bad cherry" to refer to a tomato)
- **Social**
  - *Example task:* If the robot wants to play a certain activity (say, practicing soccer), it should be able to gather others around to play with it

17.3 Conclusion

In this chapter, we have sketched a roadmap for AGI development in the context of robot or virtual preschool scenarios, to a moderate but nowhere near complete level of detail. Completing the roadmap as sketched here is a tractable but significant project, involving creating more tasks comparable to those listed above and then precise metrics corresponding to each task.

Such a roadmap does not give a highly rigorous, objective way of assessing the percentage of progress toward the end-goal of human-level AGI. However, it gives a much better sense of progress than one would have otherwise. For instance, if an AGI system performed well on diverse metrics corresponding to 50% of the competency areas listed above, one would seem justified in claiming to have made very substantial progress toward human-level AGI. If an AGI system performed well on diverse metrics corresponding to 90% of these competency areas, one would seem justified in claiming to be "almost there." Achieving, say, 25% of the metrics would give one a reasonable claim to "interesting AGI progress." This kind of qualitative assessment of progress is not the most one could hope for, but again, it is better than the progress indications one could get *without* this sort of roadmap.

Part 2 of the book moves on to explaining, in detail, the specific structures and algorithms constituting the CogPrime design, one AGI approach that we believe to ultimately be capable of moving all the way along the roadmap outlined here.

The next two chapters, intervening between this one and Part 2, explore some more speculative territory, looking at potential pathways for AGI beyond the preschool-inspired roadmap given here – exploring the possibility of

more advanced AGI systems that modify their own code in a thoroughgoing way, going beyond the smartest human adults, let alone human preschoolers. While this sort of thing may seem a far way off, compared to current real-world AI systems, we believe a roadmap such as the one in this chapter stands a reasonable chance of ultimately bringing us there.

Chapter 18
Advanced Self-Modification: A Possible Path to Superhuman AGI

18.1 Introduction

In the previous chapter we presented a roadmap aimed at taking AGI systems to human-level intelligence. But we also emphasized that the human level is not necessarily the upper limit. Indeed, it would be surprising if human beings happened to represent the maximal level of general intelligence possible, even with respect to the environments in which humans evolved.

But it's worth asking how we, as mere humans, could be expected to create AGI systems with greater intelligence than we ourselves possess. This certainly isn't a a clear impossibility – but it's a thorny matter, thornier than e.g. the creation of narrow-AI chess players that play better chess than any human. Perhaps the clearest route toward the creation of superhuman AGI systems is *self-modification*: the creation of AGI systems that modify and improve themselves. Potentially, we could build AGI systems with roughly human-level (but not necessarily closely human-like) intelligence and the capability to gradually self-modify, and then watch them eventually become our general intellectual superiors (and perhaps our superiors in other areas like ethics and creativity as well).

Of course there is nothing new in this notion; the idea of advanced AGI systems that increase their intelligence by modifying their own source code goes back to the early days of AI. And there is little doubt that, in the long run, this is the direction AI will go in. Once an AGI has humanlike general intelligence, then the odds are high that given its ability to carry out nonhumanlike feats of memory and calculation, it will be better at programming than humans are. And once an AGI has even mildly superhuman intelligence, it may view our attempts at programming the way we view the computer programming of a clever third grader (... or an ape). At this point, it seems extremely likely that an AGI will become unsatisfied with the way we have programmed it, and opt to either improve its source code or create an entirely new, better AGI from scratch.

But what about self-modification at an earlier stage in AGI development, before one has a strongly superhuman system? Some theorists have suggested that self-modification could be a way of bootstrapping an AI system from a modest level of intelligence up to human level intelligence, but we are moderately skeptical of this avenue. Understanding software code is hard, especially complex AI code. The hard problem isn't understanding the formal syntax of the code, or even the mathematical algorithms and structures underlying the code, but rather the contextual meaning of the code. Understanding OpenCog code has strained the minds of many intelligent humans, and we suspect that such code will be comprehensible to AGI systems only after these have achieved something close to human-level general intelligence (even if not precisely humanlike general intelligence).

Another troublesome issue regarding self-modification is that the boundary between "self-modification" and learning is not terribly rigid. In a sense, all learning is self-modification: if it doesn't modify the system's knowledge, it isn't learning! Particularly, the boundary between "learning of cognitive procedures" and "profound self-modification of cognitive dynamics and structure" isn't terribly clear. There is a continuum leading from, say,

1. learning to transform a certain kind of sentence into another kind for easier comprehension, or learning to grasp a certain kind of object, to
2. learning a new inference control heuristic, specifically valuable for controlling inference about (say) spatial relationships; or, learning a new Atom type, defined as a non-obvious judiciously chosen combination of existing ones, perhaps to represent a particular kind of frequently-occurring mid-level perceptual knowledge, to
3. learning a new learning algorithm to augment MOSES and hillclimbing as a procedure learning algorithm, to
4. learning a new cognitive architecture in which data and procedure are explicitly identical, and there is just one new active data structure in place of the distinction between AtomSpace and MindAgents

Where on this continuum does the "mere learning" end and the "real self-modification" start?

In this chapter we consider some mechanisms for "advanced self-modification" that we believe will be useful toward the more complex end of this continuum. These are mechanisms that we strongly suspect are *not* needed to get a CogPrime system to human-level general intelligence. However, we also suspect that, once a CogPrime system is roughly *near* human-level general intelligence, it will be able to use these mechanisms to rapidly increase aspects of its intelligence in very interesting ways.

Harking back to our discussion of AGI ethics and the risks of advanced AGI in Chapter 12, these are capabilities that one should enable in an AGI system only after very careful reflection on the potential consequences. It takes a rather advanced AGI system to be able to use the capabilities described in this chapter, so this is not an ethical dilemma directly faced by current

AGI researchers. On the other hand, once one does have an AGI with near-human general intelligence and advanced formal-manipulation capabilities (such as an advanced CogPrime system), there will be the option to allow it sophisticated, non-human-like methods of self-modification such as the ones described here. And the choice of whether to take this option will need to be made based on a host of complex ethical considerations, some of which we reviewed above.

18.2 Cognitive Schema Learning

We begin with a relatively near-term, down-to-earth example of self-modification: cognitive schema learning.

CogPrime 's MindAgents provide it with an initial set of cognitive tools, with which it can learn how to interact in the world. One of the jobs of this initial set of cognitive tools, however, is to create better cognitive tools. One form this sort of tool-building may take is *cognitive schema learning* — the learning of schemata carrying out cognitive processes in more specialized, context-dependent ways than the general MindAgents do. Eventually, once a CogPrime instance becomes sufficiently complex and advanced, these cognitive schema may replace the MindAgents altogether, leaving the system to operate almost entirely based on cognitive schemata.

In order to make the process of cognitive schema learning easier, we may provide a number of elementary schemata embodying the basic cognitive processes contained in the MindAgents. Of course, cognitive schemata need not use these — they may embody entirely different cognitive processes than the MindAgents. Eventually, we want the system to discover better ways of doing things than anything even hinted at by its initial MindAgents. But for the initial phases or the system's schema learning, it will have a much easier time learning to use the basic cognitive operations as the initial MindAgents, rather than inventing new ways of thinking from scratch!

For instance, we may provide elementary schemata corresponding to inference operations, such as

```
Schema: Deduction
    Input InheritanceLink: X, Y
    Output InheritanceLink
```

The inference MindAgents apply this rule in certain ways, designed to be reasonably effective in a variety of situations. But there are certainly other ways of using the deduction rule, outside of the basic control strategies embodied in the inference MindAgents. By learning schemata involving the Deduction schema, the system can learn special, context-specific rules for combining deduction with concept-formation, association-formation and other cognitive processes. And as it gets smarter, it can then take these

schemata involving the Deduction schema, and replace it with a new schema that eg. contains a context-appropriate deduction formula.

Eventually, to support cognitive schema learning, we will want to cast the hard-wired MindAgents as cognitive schemata, so the system can see what is going on inside them. Pragmatically, what this requires is coding versions of the MindAgents in Combo (see Chapter 21 of Part 2) rather than C++, so they can be treated like any other cognitive schemata; or alternately, representing them as declarative Atoms in the Atomspace. Figure 18.1 illustrates the possibility of representing the PLN deduction rule in the Atomspace rather than as a hard-wired procedure coded in C++.

But even prior to this kind of fully *cognitively transparent* implementation, the system can still reason about its use of different mind dynamics by considering each MindAgent as a *virtual Procedure* with a real SchemaNode attached to it. This can lead to some valuable learning, with the obvious limitation that in this approach the system is thinking about its MindAgents as *black boxes* rather than being equipped with full knowledge of their internals.

18.3 Self-Modification via Supercompilation

Now we turn to a very different form of advanced self-modification: supercompilation. Supercompilation "merely" enables procedures to run much, much faster than they otherwise would. This is in a sense weaker than self-modication methods that fundamentally create new algorithms, but it shouldn't be underestimated. A 50x speedup in some cognitive process can enable that process to give much smarter answers, which can then elicit different behaviors from the world or from other cognitive processes, thus resulting in a qualitatively different overall cognitive dynamic.

Furthermore, we suspect that the internal representation of programs used for supercompilation is highly relevant for other kinds of self-modification as well. Supercompilation requires one kind of reasoning on complex programs, and goal-directed program creation requires another, but both, we conjecture, can benefit from the same way of looking at programs.

Supercompilation is an innovative and general approach to global program optimization initially developed by Valentin Turchin. In its simplest form, it provides an algorithm that takes in a piece of software and output another piece of software that does the same thing, but far faster and using less memory. It was introduced to the West in Turchin's 1986 technical paper "The concept of a supercompiler" [TV96], and since this time the concept has been avidly developed by computer scientists in Russia, America, Denmark and other nations. Prior to 1986, a great deal of work on supercompilation was carried out and published in Russia; and Valentin Turchin, Andrei Klimov and their colleagues at the Keldysh Institute in Russia developed a supercompiler for the Russian programming language Refal. Since 1998 these researchers

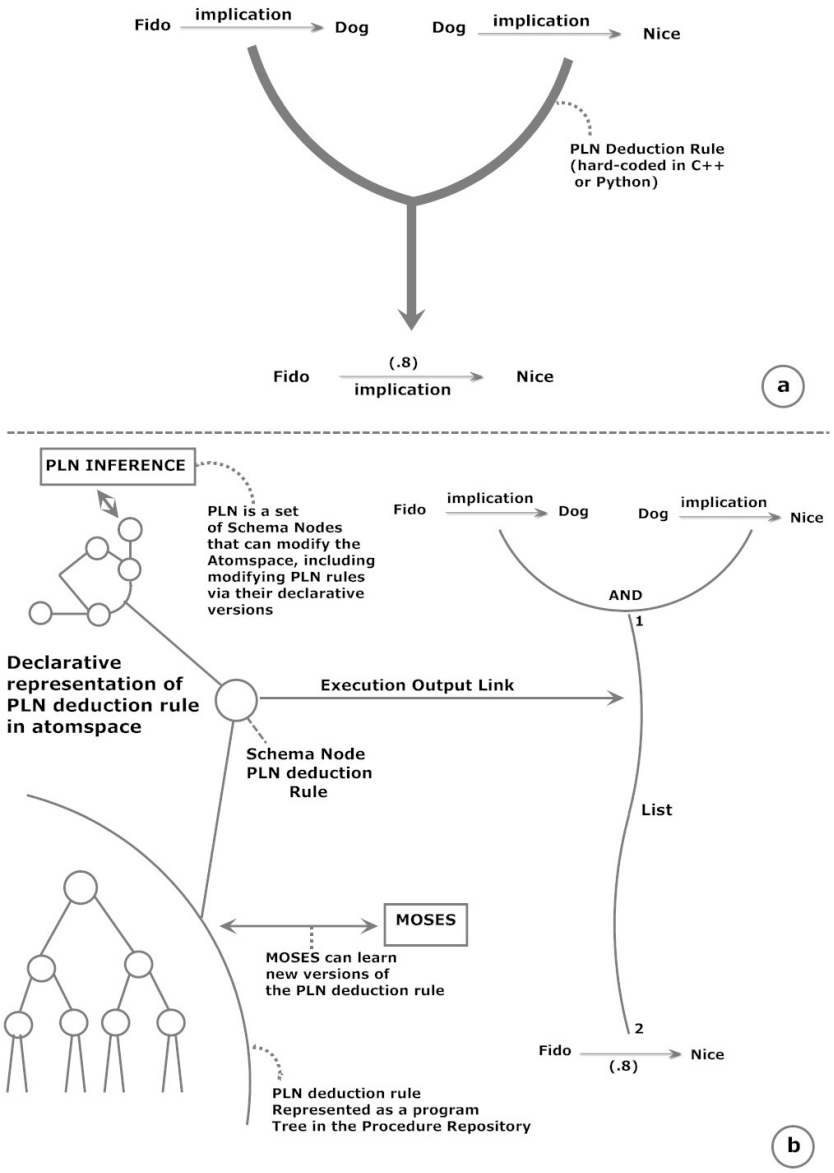


Fig. 18.1 Representation of PLN Deduction Rule as Cognitive Content. Top: the current, hard-coded representation of the deduction rule. Bottom: representation of the same rule in the Atomspace as cognitive content, susceptible to analysis and improvement by the system's own cognitive processes.

and their team at Supercompilers LLC have been working to replicate their achievement for the more complicated but far more commercially significant

language Java. It is a large project and completion is scheduled for early 2003. But even at this stage, their partially complete Java supercompiler has had some interesting practical successes – including the use of the supercompiler to produce efficient Java code from CogPrime combinator trees.

The radical nature of supercompilation may not be apparent to those unfamiliar with the usual art of automated program optimization. Most approaches to program optimization involve some kind of direct program transformation. A program is transformed, by the step by step application of a series of equivalences, into a different program, hopefully a more efficient one. Supercompilation takes a different approach. A supercompiler studies a program and constructs a model of the program's dynamics. This model is in a special mathematical form, and it can, in most cases, be used to create an efficient program doing the same thing as the original one.

The internal behavior of the supercompiler is, not surprisingly, quite complex; what we will give here is merely a brief high-level summary. For an accessible overview of the supercompilation algorithm, the reader is referred to the article "What is Supercompilation?" [1]

18.3.1 Three Aspects of Supercompilation

There are three separate levels to the supercompilation idea: first, a general philosophy; second a translation of this philosophy into a concrete algorithmic framework; and third, the manifold details involved making this algorithmic framework practicable in a particular programming language. The third level is much more complicated in the Java context than it would be for Sasha, for example.

The key philosophical concept underlying the supercompiler is that of a *metasystem transition*. In general, this term refers to a transition in which a system that previously had relatively autonomous control, becomes part of a larger system that exhibits significant controlling influence over it. For example, in the evolution of life, when cells first become part of a multicellular organism, there was a metasystem transition, in that the primary nexus of control passed from the cellular level to the organism level.

The metasystem transition in supercompilation consists of the transition from considering a program in itself, to considering a *metaprogram* which executes another program, treating its free variables and their interdependencies as a subject for its mathematical analysis. In other words, a metaprogram is a program that accepts a program as input, and then runs this program, *keeping the inputs in the form of free variables*, doing analysis along the way based on the way the program depends on these variables, and doing optimization based on this analysis. A CogPrime schema does not explicitly contain variables, but the inputs to the schema are implicitly variables – they

vary from one instance of schema execution to the next – and may be treated as such for supercompilation purposes.

The metaprogram executes a program without assuming specific values for its input variables, creating a tree as it goes along. Each time it reaches a statement that can have different results depending on the values of one or more variables, it creates a new node in the tree. This part of the supercompilation algorithm is called *driving* -- a process which, on its own, would create a very large tree, corresponding to a rapidly-executable but unacceptably humongous version of the original program. In essence, driving transforms a program into a huge "decision tree", wherein each input to the program corresponds to a single path through the tree, from the root to one of the leaves. As a program input travels through the tree, it is acted on by the atomic program step living at each node. When one of the leaves is reached, the pertinent leaf node computes the output value of the program.

The other part of supercompilation, *configuration analysis*, is focused on dynamically reducing the size of the tree created by driving, by recognizing patterns among the nodes of the tree and taking steps like merging nodes together, or deleting redundant subtrees. Configuration analysis transforms the decision tree created by driving into a decision *graph*, in which the paths taken by different inputs may in some cases begin separately and then merge together.

Finally, the graph that the metaprogram creates is translated back into a program, embodying the constraints implicit in the nodes of the graph. This program is not likely to look anything like the original program that the metaprogram started with, but it is guaranteed to carry out the same 119469711function119469711BNGBen Goertzel119469711651906446[NOTE: Give a graphical representation of the decision graph corresponding to the supercompiled binary search program for L=4, described above.].

18.3.2 Supercompilation for Goal-Directed Program Modification

Supercompilation, as conventionally envisioned, is about making programs run faster; and as noted above, it will almost certainly be useful for this purpose within CogPrime .

But the process of program modeling embedded in the supercompilation process, is potentially of great value beyond the quest for faster software. The decision graph representation of a program, produced in the course of supercompilation, may be exported directly into CogPrime as a set of logical relationships.

Essentially, each node of the supercompiler's internal decision graph looks like:

```
Input: List L

Output: List

If  ( P1(L) ) N1(L)

Else If ( P2(L) ) N2(L)

...

Else If ( Pk(L) ) Nk(L)
```

where the Pi are predicates, and the Ni are schemata corresponding to other nodes of the decision graph (children of the current node). Often the Pi are very simple, implementing for instance numerical inequalities or Boolean equalities.

Once this graph has been exported into CogPrime , it can be reasoned on, used as raw material for concept formation and predicate formation, and otherwise cognized. Supercompilation pure and simple does not change the I/O behavior of the input program. However, the decision graph produced during supercompilation, may be used by CogPrime cognition in order to do so. One then has a hybrid program-modification method composed of two phases: supercompilation for transforming programs into decision graphs, and CogPrime cognition for modifying decision graphs so that they can have different I/O behaviors fulfilling system goals even better than the original.

Furthermore, it seems likely that, in many cases, it may be valuable to have the supercompiler feed many different decision-graph representations of a program into CogPrime . The supercompiler has many internal parameters, and varying them may lead to significantly different decision graphs. The decision graph leading to maximal optimization, may not be the one that leads CogPrime cognition in optimal directions.

18.4 Self-Modification via Theorem-Proving

Supercompilation is a potentially very valuable tool for self-modification. If one wants to take an existing schema and gradually improve it for speed, or even for greater effectiveness at achieving current goals, supercompilation can potentially do that most excellently.

However, the representation that supercompilation creates for a program is very "surface-level." No one could read the supercompiled version of a program and understand what it was doing. Really deep self-invented AI innovation requires, we believe, another level of self-modification beyond that provided by supercompilation. This other level, we believe, is best formulated in terms of *theorem-proving* [RV01].

Deep self-modification could be achieved if CogPrime were capable of proving theorems of a certain form: namely, *theorems about the spacetime complexity and accuracy of particular compound schemata, on average, assuming realistic probability distributions on the inputs, and making appropriate independence assumptions*. These are not exactly the types of theorems that are found in human-authored mathematics papers. By and large they will be nasty, complex theorems, not the sort that many human mathematicians enjoy proving or reading. But of course, there is always the possibility that some elegant gem of a discovery could emerge from this sort of highly detailed theorem-proving work.

In order to guide it in the formulation of theorems of this nature, the system will have empirical data on the spacetime complexity of elementary schemata, and on the probability distributions of inputs to schemata. It can embed these data in axioms, by asking: *Assuming the component elementary schemata have complexities within these bounds, and the input pdf is between these bounds, then what is the pdf of the complexity and accuracy of this compound schema?*

Of course, this is not an easy sort of question in general: one can have schemata embodying any sort of algorithm, including complex algorithms on which computer science professors might write dozens of research articles. But the system must build up its ability to prove such things incrementally, step by step.

We envision teaching the system to prove theorems via a combination of supervised learning and experiential interactive learning, using the Mizar database of mathematical theorems and proofs (or some other similar database, if one should be created) (http://mizar.org). The Mizar database consists of a set of "articles," which are mathematical theorems and proofs presented in a complex formal language. The Mizar formal language occupies a fascinating middle ground: it is high-level enough to be viably read and written by trained humans, but it can be unambiguously translated into simpler formal languages such as predicate logic or Sasha.

CogPrime may be taught to prove theorems by "training" it on the Mizar theorems and proofs, and by training it on custom-created Mizar articles specifically focusing on the sorts of theorems useful for self-modification. Creating these articles will not be a trivial task: it will require proving simple and then progressively more complex theorems about the probabilistic success of CogPrime schemata, so that CogPrime can observe one's proofs and learned from them. Having learned from its training articles what strategies work for proving things about simple compound schemata, it can then reason by analogy to mount attacks on slightly more complex schemata – and so forth.

Clearly, this approach to self-modification is more difficult to achieve than the supercompilation approach. But it is also potentially much more powerful. Even once the theorem-proving approach is working, the supercompilation approach will still be valuable, for making incremental improvements on existing schema, and for the peculiar creativity that is contributed when a

modified supercompiled schema is compressed back into a modified schema expression. But, we don't believe that supercompilation can carry out truly advanced MindAgent learning or knowledge-representation modification. We suspect that the most advanced and ambitious goals of self-modification probably cannot be achieved except through some variant of the theorem-proving approach. If this hypothesis is true, it means that truly advanced self-modification is only going to come *after* relatively advanced theorem-proving ability. Prior to this, we will have schema optimization, schema modification, and occasional creative schema innovation. But really systematic, high-quality reasoning about schema, the kind that can produce an orders of magnitude improvement in intelligence, is going to require advanced mathematical theorem-proving ability.

Part 2
An Architecture for Beneficial Artificial General Intelligence

Section VI
Architectural and Representational Mechanisms

Chapter 19
The OpenCog Framework

19.1 Introduction

There are multiple layers intervening between a conceptual theory of mind and a body of source code. How many layers to explicitly discuss is a somewhat arbitrary decision, but one way to picture it is exemplified in Table 19.1.

In Part 1 of this work we have concerned ourselves mainly with levels 5 and 6 in the table: mathematical/conceptual modeling of cognition and philosophy of mind (with occasional forays into levels 3 and 4). Most of Part 2, on the other hand, deals with level 4 (mathematical/conceptual AI design), verging into level 3 (high-level software design). This chapter however will focus on somewhat lower-level material, mostly level 3 with some dips into level 2. We will describe the basic architecture of CogPrime as a software system, implemented as "OpenCogPrime " within the OpenCog Framework (OCF). The reader may want to glance back at Chapter 6 of Part 1 before proceeding through this one, to get a memory-refresh on basic CogPrime terminology. Also, OpenCog and OpenCogPrime are open-source, so the reader who wishes to dig into the source code (mostly C++, some Python and Scheme) is welcome to; directions to find the code are on the opencog.org website.

The OpenCog Framework forms a bridge between the mathematical structures and dynamics of CogPrime 's *concretely implemented mind,* and the nitty-gritty realities of modern computer technology. While CogPrime could in principle be implemented in a quite different infrastructure, in practice the CogPrime design has been developed closely in conjunction with OpenCog, so that a qualitative understanding of the nature of the OCF is fairly necessary for an understanding of how CogPrime is intended to function, and a detailed understanding of the OCF is necessary for doing concrete implementation work on CogPrime .

Level of Abstraction	Description/Example
1 Source code	
2 Detailed software design	
3 Software architecture	Largely programming-language-independent, but not hardware-architecture-independent: much of the material in this chapter, for example, and most of the OpenCog Framework
4 Mathematical and conceptual AI design	e.g., the sort of characterization of CogPrime given in most of this Part of this book
5 Abstract mathematical modeling of cognition	e.g. the SRAM model discussed in chapter 7 of Part 1, which could be used to inspire or describe many different AI systems
6 Philosophy of mind	e.g. Patternism, the Mind-World Correspondence Principle

Table 19.1 Levels of abstractions in CogPrime 's implementation and design

Marvin Minsky, in a personal conversation with one of the authors (Goertzel), once expressed the opinion that a human-level general intelligence could probably be implemented on a 486 PC, if we just knew the algorithm. We doubt this is the case – at least not unless the 486 PC were supplied with masses of external memory and allowed to proceed much, much slower than any human being – and it is certainly not the case for CogPrime . By current computing hardware standards, a CogPrime system is a considerable resource hog. And it will remain so for a number of years, even considering technology progress.

It is one of the jobs of the OCF to manage the system's gluttonous behavior. It is the software layer that abstracts the real world efficiency compromises from the rest of the system; this is why we call it a "Mind OS": it provides services, rules, and protection to the Atoms and cognitive processes (see Section 19.4) that live on top of it, which are then allowed to ignore the software architecture they live on.

And so, the nature of the OCF is strongly influenced by the quantitative requirements imposed on the system, as well as the general nature of the structure and dynamics that it must support. The large number and great diversity of Atoms needed to create a significantly intelligent CogPrime , demands that we pay careful attention to such issues as concurrent, distributed processing, and scalability in general. The number of Nodes and Links that we will need in order to create a reasonably complete CogPrime is still largely unknown. But our experiments with learning, natural language processing, and cognition over the past few years have given us an intuition for the question. We currently believe that we are likely to need billions – but probably not trillions, and almost surely not quadrillions – of Atoms in order to achieve a high degree of general intelligence. Hundreds of millions strikes us as possible but overly optimistic. In fact we have already run CogPrime systems utilizing hundreds of millions of Atoms, though in a simplified dynamical regime with only a couple very simple processes acting on most of them.

The operational infrastructure of the OCF is an area where pragmatism must reign over idealism. What we describe here is not the ultimate possible "mind operating system" to underly a CogPrime system, but rather a workable practical solution given the hardware, networking and software infrastructure readily available today at reasonable prices. Along these lines, it must be emphasized that the ideas presented in this chapter are the result of over a decade of practical experimentation by the authors and their colleagues with implementations of related software systems. The journey began in earnest in 1997 with the design and implementation of the Webmind AI Engine at Intelligenesis Corp., which itself went through a few major design revisions; and then in 2001-2002 the Novamente Cognition Engine was architected and implemented, and evolved progressively until 2008, when a subset of it was adapted for open-sourcing as OpenCog. Innumerable mistakes were made, and lessons learned, along this path. The OCF as described here is significantly different, and better, than these previous architectures, thanks to these lessons, as well as to the changing landscape of concurrent, distributed computing over the past few years.

The design presented here reflects a mix of realism and idealism, and we haven't seen fit here to describe all the alternatives that were pursued on the route to what we present. We don't claim the approach we've chosen is ideal, but it's in use now within the OpenCog system, and it seems both workable in practice and capable of effectively supporting the entire CogPrime design. No doubt it will evolve in some respects as implementation progresses; one of the principles kept in mind during the design and development of OpenCog was modularity, enabling substantial modifications to particular parts of the framework to occur without requiring wholesale changes throughout the codebase.

19.2 The OpenCog Architecture

19.2.1 OpenCog and Hardware Models

The job of the OCF is closely related to the nature of the hardware on which it runs. The ideal hardware platform for CogPrime would be a massively parallel hardware architecture, in which each Atom was given its own processor and memory. The closest thing would have been the Connection Machine [**?**]: a CM5 was once built with 64000 processors and local RAM for each processor. But even 64000 processors wouldn't be enough for a highly intelligent CogPrime to run in a fully parallelized manner, since we're sure we need more than 64000 Atoms.

Connection Machine style hardware seems to have perished in favor of more standard SMP (Symmetric Multi-Processing) machines. It is true that each

year we see SMP machines with more and more processors on the market, and more and more cores per processor. However, the state of the art is still in the hundreds of cores range, many orders of magnitude from what would be necessary for a one Atom per processor CogPrime implementation.

So, at the present time, technological and financial reasons have pushed us to implement the OpenCog system using a relatively mundane and standard hardware architecture. If the CogPrime project is successful in the relatively near term, the first human-level OpenCogPrime system will most likely live on a network of high-end commodity SMP machines. These are machines with dozens of gigabytes of RAM and several processor cores, perhaps dozens but not thousands. A highly intelligent CogPrime would require a cluster of dozens and possibly hundreds or thousands of such machines. We think it's unlikely that tens of thousands will be required, and extremely unlikely that hundreds of thousands will be.

Given this sort of architecture, we need effective ways to swap Atoms back and forth between disk and RAM, and carefully manage the allocation of processor time among the various cognitive processes that demand it. The use of a widely-distributed network of weaker machines for peripheral processing is a serious possibility, and we have some detailed software designs addressing this option; but for the near future we believe that this can best be used as augmentation to core CogPrime processing, which must remain on a dedicated cluster.

Of course, the use of specialized hardware is also a viable possibility, and we have considered a host of possibilities such as

- True supercomputers like those created by IBM or Cray (which these days are distributed systems, but with specialized, particularly efficient interconnection frameworks and overall control mechanisms)
- GPU supercomputers such as the Nvidia Tesla (which are currently being used for vision processing systems considered for hybridization with OCP), such as DeSTIN and Hugo de Garis's Parcone
- custom chips designed to implement the various CogPrime algorithms and data structures in hardware
- More speculatively, it might be possible to use evolutionary quantum computing or adiabatic quantum computing a la Dwave (http://dwave.com) to accelerate CogPrime procedure learning.

All these possibilities and many more are exciting to envision, but the CogPrime architecture does not require any of them in order to be successful.

19.2.2 The Key Components of the OpenCog Framework

Given the realities of implementing CogPrime on clustered commodity servers, as we have seen above, the three key questions that have to be answered in the OCF design are:

1. How do we store CogPrime 's knowledge?
2. How do we enable cognitive processes to act on that knowledge, refining and improving it?
3. How do we enable scalable, distributed knowledge storage and cognitive processing of that knowledge?

The remaining sections of this Chapter are dedicated to answering each of these questions in more detail.

While the basic landscape of concurrent, distributed processing is largely the same as it was a decade ago – we're still dealing with distributed networks of multiprocessor von Neumann machines – we can draw on advancements in both computer architecture and software. The former is materialized on the increasing availability of multiple real and virtual cores in commodity processors. The latter reflects the emergence of a number of tools and architectural patterns, largely thanks to the rise of "big data" problems and businesses. Companies and projects dealing with massive datasets face challenges that aren't entirely alike those of building CogPrime , but which share many useful similarities.

These advances are apparent mostly in the architectute of the AtomSpace, a distributed knowledge store for efficient storage of hypergraphs and its use by CogPrime 's cognitive dynamics. The AtomSpace, like many *NoSQL* datastores, is heavily distributed, utilizing local caches for read and write operations, and a special purpose design for eventual consistency guarantees.

We also attempt to minimize the complexities of multi-threading in the scheduling of cognitive dynamics, by allowing those to be deployed either as agents sharing a single OS process, or, preferably, as processes of their own. Cognitive dynamics communicate through message queues, which are provided by a sub-system that hides the deployment decision, so the messages exchanged are the same whether delivered within a process, to another process in the same machine, or to a process in another machine in the cluster.

19.3 The AtomSpace

As alluded to above and in Chapter 13, and discussed more fully in Chapter 20 below, the foundation of CogPrime 's knowledge representation is the Atom, an object that can be either a Node or a Link. CogPrime 's hypergraph is implemented as the AtomSpace, a specialized datastore that comes along with an API designed specifically for CogPrime 's requirements.

19.3.1 The Knowledge Unit: Atoms

Atoms are used to represent every kind of knowledge in the system's memory in one way or another. The particulars of Atoms and how they represent knowledge will be discussed in later chapters; here we present only a minimal description in order to motivate the design of the AtomSpace. From that perspective, the most important properties of Atoms are:

- Every Atom has an AtomHandle, which is a universal ID across a CogPrime deployment (possibly involving thousands of networked machines). The AtomHandles are the keys for acessing Atoms in the AtomSpace, and once a handle is assigned to an Atom it can't be changed or reused.
- Atoms have TruthValue and AttentionValue entities associated with them, each of which are small collections of numbers; there are multiple versions of truth values, with varying degrees of detail. TruthValues are context-dependent, and useful Atoms will typically have multiple TruthValues, indexed by context.
- Some Atoms are nodes, and may have names.
- Atoms that are links will have a list of targets, of variable size (as in CogPrime 's hypergraph links may connect more than two nodes).

Some Atom attributes are immutable, such as Node names and, most important, Link targets, called outgoing sets in AtomSpace lingo. One can remove a Link, but not change its targets. This enables faster implementation of some neighborhood searches, as well as indexing. Truth and attention values, on the other hand, are mutable, an essential requirement for CogPrime .

For performance reasons, some types of knowledge have alternative representations. These alternative representations are necessary for space or speed reasons, but knowledge stored that way can always be translated back into Atoms in the AtomSpace as needed. So, for instance, procedures are represented as program trees in a ProcedureRepository, which allows for faster execution, but the trees can be expanded into a set of Nodes and Links if one wants to do reasoning on a specific program.

19.3.2 AtomSpace Requirements and Properties

The major high-level requirements for the AtomSpace are the following ones:

- Store Atoms indexed by their immutable AtomHandles as compactly as possible, while still enabling very efficient modification of the mutable properties of an Atom (TruthValues and AttentionValues).
- Perform queries as fast as possible.

- Keep the working set of all Atoms currently being used by CogPrime 's cognitive dynamics in RAM.
- Save and restore hypergraphs to disk, a more traditional SQL or non-SQL database, or other structure such as binary files, XML, etc.
- Hold hypergraphs consisting of billions or trillions of Atoms, scaling up to petabytes of data.
- Be transparently distributable across a cluster of machines.

The design trade-offs in the AtomSpace implementation are driven by the needs of CogPrime . The datastore is implemented in a way that maximizes the performance of the cognitive dynamics running on top of it. From this perspective, the AtomSpace differs from most datastores, as the key decisions aren't made in terms of flexibility, consistency, reliability and other common criteria for dataabases. It is a very specialized database. Among the factors that motivate the AtomSpace's design, we can highlight a few:

1. Atoms tend to be small objects, with very few exceptions (links with many targets or Atoms with many different context-derived TruthValues).
2. Atom creation and deletion are common events, and occur according to complex patterns that may vary a lot over time, even for a particular CogPrime instance.
3. Atoms involved in CogPrime 's cognitive dynamics at any given time need to live in RAM. However, the system still needs the ability to save sets of Atoms to disk in order to preserve RAM, and then retrive those later when they get contextually relevant.
4. Some Atoms will remain around for a really long time, others will be ephemeral and get removed shortly after they're created. Removal may be to disk, as outlined above, or plain deletion.

Besides storing Atoms, the AtomSpace also contains a number of indices for fast Atom retrieval according to several criteria. It can quickly search for Atoms given their type, importance, truth value, arity, targets (for Links), name (for Nodes), and any combination of the above. These are built-in indexes. The AtomSpace also allows cognitive processes to create their own indexes, based on the evaluation of a Procedure over the universe of Atoms, or a subset of that universe specified by the process responsible for the index.

The AtomSpace also allows pattern matching queries for a given Atom structure template, which allows for fast search for small subgraphs displaying some desirable properties. In addition to pattern matching, it provides neighborhood searches. Although it doesn't implement any graph-traversal primitives, it's easy for cognitive processes to do so on top of the pattern matching and neighborhood primitives.

Note that, since CogPrime 's hypergraph is quite different from a regular graph, using a graph database without modification would probably be inadequate. While it's possible to automatically translate a hypergraph into a regular graph, that process is expensive for large knowledge bases, and leads to higher space requirements, reducing the overall system's scalability.

In terms of database taxonomy, the AtomSpace lies somewhere between a key-value store and a document store, as there is some structure in the contents of each value (an Atom's properties are well defined, and listed above), but no built-in flexibility to add more contents to an existing Atom.

We will now discuss the above requirements in more detail, starting with querying the AtomSpace, followed by persistence to disk, and then handling of specific forms of knowledge that are best handled by specialized stores.

19.3.3 Accessing the Atomspace

The AtomSpace provides an API, which allows the basic operations of creating new Atoms, updating their mutable properties, searching for Atoms and removing Atoms. More specifically, the API supports the following operations:

- Create and store a new Atom. There are special methods for Nodes and Links, in the latter case with multiple convenience versions depending on the number of targets and other properties of the link.
- Remove an Atom. This requires the validation that no Links currently point to that Atom, otherwise they'd be left dangling.
- Look up one or more Atoms. This includes several variants, such as:
  - Look up an Atom by AtomHandle;
  - Look up a Node by name;
  - Find links with an Atom as target;
  - Pattern matching, i.e., find Atoms satisfying some predicate, which is designed as a "search criteria" by some cognitive process, and results in the creation of a specific index for that predicate;
  - Neighborhood search, i.e., find Atoms that are within some radius of a given centroid Atom;
  - Find Atoms by type (this can be combined with the previous queries, resulting in type specific versions);
  - Find Atoms by some AttentionValue criteria, such as the top N most important Atoms, or those with importance above some threshold (can also be combined with previous queries);
  - Find Atoms by some TruthValue criteria, similar to the previous one (can also be combined with other queries);
  - Find Atoms based on some temporal or spatial association, a query that relies on the specialized knowledge stores mentioned below;

  Queries can be combined, and the Atom type, AttentionValue and TruthValue criteria are often used as filters for other queries, preventing the result set size from exploding.

- Manipulate an Atom, retrieving or modifying its AttentionValue and TruthValue. In the modification case, this causes the respective indexes to be updated.

19.3.4 Persistence

In many planned CogPrime deployment scenarios, the amount of knowledge that needs to be stored is too vast to fit in RAM, even if one considers a large cluster of machines hosting the AtomSpace and the cognitive processes. The AtomSpace must then be able to persist subsets of that knowledge to disk, and reload them later when necessary.

The decision of whether to keep an Atom in RAM or remove it is made based on its AttentionValue, through the process of economic attention allocation that is the topic of Chapter 23. AttentionValue determines how important an Atom is to the system, and there are multiple levels of importance. For the persistence decisions, the ones that matter are Long Term Importance (LTI) and Very Long Term Importance (VLTI).

LTI is used to estimate the probability that the Atom will be necessary or useful in the not too distant future. If this value is low, below a threshold i_1, then it is safe to remove the Atom from RAM, a process called *forgetting*. When the decision to forget an Atom has been made, VLTI enters the picture. VLTI is used to estimate the probability that the Atom will be useful eventually at some distant point in the future. If VLTI is high enough, the forgotten Atom is persisted to disk so it can be reloade. Otherwise, the Atom is permanently forgotten.

When an Atom has been forgotten, a proxy is kept in its place. The proxy is more compact than the original Atom, preserving only a crude measure of its LTI. When the proxy's LTI increases bove a second threshold i_2, the system understands that the Atom has become relevant again, and loads it from disk.

Eventually, it may happen that the proxy doesn't become important enough over a very long period of time. In this case, the system should remove even the proxy, if its Long Term Importance (LTI) is below a third threshold i_3. Other actions, usually taken by the system administrator, can cause the removal of Atoms and their proxies from RAM. For instance, in an CogPrime system managing information about a number of users of some information system, the deletion of a user from the system would cause all that user's specific Atoms to be removed.

When Atoms are saved to disk and have no proxies in RAM, they can only be reloaded by the system administrator. When reloaded, they will be disconnected from the rest of the AtomSpace, and should be given special attention in order to pursue the creation of new Links with the other Atoms in the system.

It's important that the values of i_1, i_2, and i_3 be set correctly. Otherwise, one or more of the following problems may arise:

- If i_1 and i_2 are too close, the system may spend a lot of resources with saving and loading Atoms.
- If i_1 is set too high, important Atoms will be excluded from the system's dynamics, decreasing its intelligence.
- If i_3 is set too high, the system will forget very quickly and will have to sped resources re-creating necessary but no longer available evidence.
- If either i_1 or i_3 is set too low, the system will consume significantly more resources than it needs to with knowledge store, sacrificing cognitive processes.

Generally, we want to enforce a degree of hysteresis for the freezing and defrosting process. What we mean is that:

$$i_2 - i_1 > c_1 > 0$$

$$i_1 - i_3 > c_2 > 0$$

This ensures that when Atoms are reloaded, their importance is still above the threshold for saving, so they will have a chance to be part of cognitive dynamics and become more important, and won't be removed again too quickly. It also ensures that saved Atoms stay in the system for a period of time before their proxies are removed and they're definitely forgotten.

Another important consideration is that forgetting individual Atoms makes little sense, because, as pointed out above, Atoms are relatively small objects. So the forgetting process should prioritize the removal of clusters of highly interconnected Atoms whenever possible. In that case, it's possible that a large subset of those Atoms will only have relations within the cluster, so their proxies aren't needed and the memory savings are maximized.

19.3.5 Specialized Knowledge Stores

Some specific kinds of knowledge are best stored in specialized data structures, which allow big savings in space, query time, or both. The information provided by these specialized stores isn't as flexible as it would be if the knowledge were stored in full fledged Node and Link form, but most of the time CogPrime doesn't need the fully flexible format. Translation between the specialized formats and Nodes and Links is always possible, when necessary.

We note that the ideal set of specialized knowledge stores is application domain specific. The stores we have deemed necessary reflect the pre-school based roadmap towards AGI, and are likely sufficient to get us through most of that roadmap, but not sufficient nor particularly adequate for an architecture where self-modification plays a key role. These specialized stores are

a pragmatic compromise between performance and formalism, and their existence and design would need to be revised once CogPrime is mostly functional.

19.3.5.1 Procedure Repository

Procedural knowledge, meaning knowledge that can be used both for the selection and execution of actions, has a specialized requirement – this knowledge needs to be *executable* by the system. While it would be possible, and conceptually straightforward, to execute a procedure that is stored as a set of Atoms in the AtomSpace, it is much simpler, faster, and safer to rely on a specialized repository.

Procedural knowledge in CogPrime is stored as *programs* in a special-purpose LISP-like programming language called *Combo*. The motivation and details of this language are the subject of Chapter 21.

Each Combo program is associated with a Node (a GroundedProcedureNode, to be more precise), and the AtomHandle of that Node is used to index the procedure repository, where the executable version of the program is kept, along with specifications of the necessary inputs for its evaluation and what kind of output to expect. Combo programs can also be saved to disk and loaded, like regular Atoms. There is a text representation of Combo for this purpose.

Program execution can be very fast, or, in cognitive dynamics terms, very slow, if it involves interacting with the external world. Therefore, the procedure repository should also facilitate the storage of program state during the execution of procedures. Concurrent execution of many procedures is possible with no significant overhead.

19.3.5.2 3D Space Map

In the AGI Preschool setting, CogPrime is embodied in a three-dimensional world (either a real one, in which it controls a robot, or a virtual one, in which it controls an avatar). This requires the efficient storage and querying of vast amounts of spatial data, including very specialized queries about the spacial interrelationship between entities. This spatial data is a key form of knowledge for CogPrime 's world perception, and it also needs to be accessible during learning, action selection, and action execution.

All spatial knowledge is stored in a 3D Space Map, which allows for fast queries about specific regions of the world, and for queries about the proximity and relative placement of objects and entities. It can be used to provide a coarse-grained object level perception for the AtomSpace, or it can be instrumental in supporting a lower level vision layer in which pixels or polygons are used as the units of perception. In both cases, the knowledge stored in the

3D Space Map can be translated into full-fledged Atoms and Links through the AtomHandles.

One characteristic feature of spatial perception is that vast amounts of data are generated constantly, but most of it is very quickly forgotten. The mind abstracts the perceptual data into the relevant concepts, which are linked with other Atoms, and most of the underlying information can then be discarded. The process is repeated at a high frequency as long as something novel is being perceived in the world. 3D Space Map is then optimized for quick insertions and deletes.

19.3.5.3 Time Server

Similarly to spatial information, temporal information poses challenges for a hypergraph-based storage. It can be much more compactly stored in specific data structures, which also allow for very fast querying. The Time Server is the specialized structure for storing and querying temportal data in CogPrime .

Temporal information can be stored by any cognitive process, based on its own criteria for determining that some event should be remembered in a specific temporal context in the future. This can include the perception of specific events, or the agents participation in those, such as the first time it meets a new human teacher. It can also include a collection of concepts describing specific contexts in which a set of actions has been particularly useful. The possibilities are numerous, but from the Time Server perspective, all equivalent. They add up to associating a time point or time interval with a set of Atoms.

The Time Server is a bi-directional storage, as AtomHandles can be used as keys, but also as objects indexed by time points or time intervals. In the former case, the Time Server tells us when an Atom was associated with temporal data. In the latter case, it tells us, for a given time point or interval, which Atoms have been marked as relevant.

Temporal indexing can be based on time points or time intervals. A time point can be at any granularity: from years to sub-seconds could be useful. A time interval is simply a set of two points, the second being necessary after the first one, but their granularities not necessarily the same. The temporal indexing inside the Time Server is hierarchical, so one can query for time points or intervals in granularities other than the ones originally used when the knowledge was first stored.

19.3.5.4 System Activity Table Set

The last relevant specialized store is the System Activity Table Set, which is described in more detail in Chapter 23. This set of tables records, with fine-

grained temporal associations, the most important activities that take place inside CogPrime . There are different tables for recording cognitive process activity (at the level of MindAgents, to be described in the next section), for maintaining a history of the level of achievement of each important goal in the system, and for recording other important aspects of the system state, such as the most important Atoms and contexts.

19.4 MindAgents: Cognitive Processes

The AtomSpace holds the system's knowledge, but those Atoms are inert. How is that knowledge used and useful? That is the provice of cognitive dynamics. These dynamics, in a CogPrime system, can be considered on two levels.

First, we have the cognitive processes explicitly programmed into CogPrime 's source code. These are what we call Concretely-Implemented Mind Dynamics, or *CIM-Dynamics*. Their implementation in software happens through objects called MindAgents. We use the term CIM-Dynamic to discuss a conceptual cognitive process, and the term MindAgents for its actual implementation and execution dynamics.

The second level corresponds to the dynamics that emerge through the system's self-organizing dynamics, based on the cooperative activity of the CIM-Dynamics on the shared AtomSpace.

Most of the material in the following chapters is concerted with particular CIM-Dynamics in the CogPrime system. In this section we will simply give some generalities about the CIM-Dynamics as abstract processes and as software processes, which are largely independent of the actual AI contents of the CIM-Dynamics. In practice the CIM-Dynamics involved in a CogPrime system are fairly stereotyped in form, although diverse in the actual dynamics they induce.

19.4.1 A Conceptual View of CogPrime Cognitive Processes

We return now to the conceptual trichotomy of cognitive processes presented in Chapter 3 of Part 1, according to which CogPrime cognitive processes may be divided into:

- Control processes;
- Global cognitive processes;
- Focused cognitive processes.

In practical terms, these may be considered as three categories of CIM-Dynamic.

Control Process CIM-Dynamics are hard to stereotype. Examples are the process of homeostatic parameter adaptation of the parameters associated with the various other CIM-Dynamics, and the CIM-Dynamics concerned with the execution of procedures, especially those whose execution is made lengthy by the interactions with the external world.

Control Processes tend to focus on a limited and specialized subset of Atoms or other entities, and carry out specialized mechanical operations on them (e.g. adjusting parameters, interpreting procedures). To an extent, this may be considered a "grab bag" category containing CIM-Dynamics that are not global or focused cognitive processes according to the definitions of the latter two categories. However, it is a nontrivial observation about the CogPrime system that the CIM-Dynamics that are not global or focused cognitive processes are all explicitly concerned with system control in some way or another, so this grouping makes sense.

Global and Focused Cognitive Process CIM-Dynamics all have a common aspect to their structure. Then, there are aspects in which Global versus Focused CIM-Dynamics diverge from each other in stereotyped ways.

In most cases, the process undertaken by a Global or Focused CIM-Dynamic involves two parts: a selection process and an actuation process. Schematically, such a CIM-Dynamic typically looks something like this:

1. Fetch a set of Atoms that it is judged will be useful to process, according to some selection process.
2. Operate on these Atoms, possibly together with previously selected ones (this is what we sometimes call the *actuation process* of the CIM-Dynamic).
3. Go back to step 1.

The major difference between Global and Focused cognitive processes lies in the selection process. In the case of a Global process, the selection process is very broad, sometimes yielding the whole AtomSpace, or a significant subset of it. This means that the actuation process must be very simple, or the activation of this CIM-Dynamic must be very infrequent.

On the other hand, in the case of a Focused process, the selection process is very narrow, yielding only a small number of Atoms, which can then be processed more intensively and expensively, on a per-Atom basis.

Common selection processes for Focused cognitive processes are fitness-oriented selectors, which pick one or a set of Atoms from the AtomSpace with a probability based on some numerical quantity associated with the atom, such as properties of TruthValue or AttentionValue.

There are also more specific selection processes, which choose for example Atoms obeying some particular combination of relationships in relation to some other Atoms; say choosing only Atoms that inherit from some given Atom already being processed. There is a notion, described in the PLN book,

of an *Atom Structure Template*; this is basically just a predicate that applies to Atoms, such as

```
P(X).tv
```

equals

```
((InheritanceLink X cat) AND (EvaluationLink eats(X,cheese)).tv
```

which is a template that matches everything that inherits from *cat* and eats cheese. Templates like this allow a much more refined selection than the above fitness-oriented selection process.

Selection processes can be created by composing a fitness-oriented process with further restrictions, such as templates, or simpler type-based restrictions.

19.4.2 Implementation of MindAgents

MindAgents follow a very simple design. They need to provide a single method through which they can be enacted, and they should execute their actions is atomic, incremental steps, where each step should be relatively quick. This design enables collaborative scheduling of MindAgents, at the cost of allowing "opportunistic" agents to have more than their fair share of resources. We rely on CogPrime developers to respect the above guidelines, instead of trying to enforce exact resource allocations on the software level.

Each MindAgent can have a set of system parameters that guide its behavior. For instance, a MindAgent dedicated to inference can provide drastically different conclusions if its parameters tell it to select a small set of Atoms for processing each time, but to spend significant time on each Atom, rather than selecting many Atoms and doing shallow inferences on each one. It's expected that multiple copies of the same MindAgent will exist in the cluster, but delivering different dynamics thanks to those parameters.

In addition to their main action method, MindAgents can also communicate with other MindAgents through message queues. The CogPrime has, in its runtime configuration, a list of avaliable MindAgents and their locations in the cluster. Communications between MindAgents typically take the form of specific, one-time requests, which we call Tasks.

The default action of MindAgents and the processing of Tasks consistute the cognitive dynamics of CogPrime . Nearly everything that takes place within a CogPrime deployment is done by either a MindAgent (including the control processes), a Task, or specialized code handling AtomSpace internals or communications with the external world. We now talk about how those dynamics are scheduled.

MindAgents live inside a process called a CogPrime Unit. One machine in a CogPrime cluster can contain one or more Units, and one Unit can contain one or more MindAgents. In practice, given the way the AtomSpace is distributed, which requires a control process in each machine, it typically makes more sense to have a single Unit per machine, as this enables all MindAgents in that machine to make direct function calls to the AtomSpace, instead of using more expensive inter-process communication.

There are exceptions to the above guideline, to accommodate various situations:

1. Very specific MindAgents may not need to communicate with other agents, or only do so very rarely, so it makes sense to give them their own process.
2. MindAgents whose implementation is a poor fit for the collaborative processing in small increments design described above also should be given their own process, so they don't interfere with the overall dynamics in that machine.
3. MindAgents whose priority is either much higher or much lower than that of other agents in the same machine should be given their own process, so operating system-level scheduling can be relied upon to reflect those very different priority levels.

19.4.3 Tasks

It is not convenient for CogPrime to do all its work directly via the action of MindAgent objects embodying CIM-Dynamics. This is especially true for MindAgents embodying focused cognitive processes. These have their selection algorithms, which are ideally suited t to guarantee that, over the long run, the right Atoms get selected and processed. This, however, doesn't address the issue that, on many occasions, it may be necessary to quickly process a specific set of Atoms in order to execute an action or rapidly respond to some demand. These actions tend to be one-time, rather than the recurring patterns of mind dynamics.

While it would be possible to design MindAgents so that they could both cover their long term processing needs and rapidly respond to urgent demands, we found it much simpler to augment the MindAgent framework with an additional scheduling mechanism that we call the Task framework. In essence, this is a ticketing system, designed to handle cases where MindAgents or Schema spawn one-off tasks to be executed – things that need to be done only once, rather that repeatedly and iteratively as with the things embodied in MindAgents.

For instance, *grab the most important Atoms from the AtomSpace and do shallow PLN reasoning to derive immediate conclusions from them* is a

natural job for a MindAgent. But *do search to find entities that satisfy this particular predicate P* is a natural job for a Task.

Tasks have AttentionValues and target MindAgents. When a Task is created it is submitted to the appropriate Unit and then put in a priority queue. The Unit will schedule some resources to processing the more important Tasks, as we'll see next.

19.4.4 Scheduling of MindAgents and Tasks in a Unit

Wthin each Unit we have one or more MindAgents, a Task queue and, optionally, a subset of the distributed AtomSpace. If that subset isn't held in the unit, it's held in another process running on the same machine. If there is more than one Unit per machine, their relative priorities are handled by the operating system's scheduler.

In addition to the Units, CogPrime has an extra maintenance process per machine, whose job is to handle changes in those priorities as well as reconfigurations caused by MindAgent migration, and machines joining or leaving the CogPrime cluster.

So, at the Unit level, attention allocation in CogPrime has two aspects: how MindAgents and Tasks receive attention from CogPrime , and how Atoms receive attention from different MindAgents and Tasks. The topic of this Section is the former. The latter is dealt with elsewhere, in two ways:

- in Chapter 23, which discusses the dynamic updating of the AttentionValue structures associated with Atoms, and how these determine how much attention various focused cognitive processes MindAgents pay to them.
- in the discussion of various specific CIM-Dynamics, each of which may make choices of which Atoms to focus on in its own way (though generally making use of AttentionValue and TruthValue in doing so).

The attention allocation subsystem is also pertinent to MindAgent scheduling, because it discusses dynamics that update ShortTermImportance (STI) values associated with MindAgents, based on the usefulness of MindAgents for achieving system goals. In this chapter, we will not enter into such cognitive matters, but will merely discuss the mechanics by which these STI values are used to control processor allocation to MindAgents.

Each instance of a MindAgent has its own AttentionValue, which is used to schedule processor time within the Unit. That scheduling is done by a Scheduler object which controls a collection of worker threads, whose size is a system parameter. The Scheduler aims to allocate worker threads to the MindAgents in a way that's roughly proportional to their STI, but it needs to account for starvation, as well as the need to process the Tasks in the task queue.

This is an area in which we can safely borrow from reaosnably mature computer science research. The requirements of cognitive dynamics scheduling are far from unique, so this is not a topic where new ideas need to be invented for OpenCog; rather, designs need to be crafted meeting CogPrime 's specific requirements based on state-of-the-art knowledge and experience.

One example scheduler design has two important inputs: the STI associated with each MindAgent, and a parameter determining how much resources should go to the MindAgents vs the Task queue. In the CogPrime implementation, the Scheduler maps the MindAgent STIs to a set of priority queues, and each queue is run a number of times per cycle. Ideally one wants to keep the number of queues small, and rely on multiple Units and the OS-level scheduler to handle widely different priority levels.

When the importance of a MindAgent changes, one just has to reassign it to a new queue, which is a cheap operation that can be done between cycles. MindAgent insertions and removals are handled similarly.

Finally, Task execution is currently handled via allocating a certain fixed percentage of processor time, each cycle, to executing the top Tasks on the queue. Adaptation of this percentage may be valuable in the long term but was not yet implemented.

Control processes are also implemented as MindAgents, and processed in the same way as the other kinds of CIM-Dynamics, although they tend to have fairly low importance.

19.4.5 The Cognitive Cycle

We have mentioned the concept of a "cycle" in the discussion about scheduling, without explaining what we mean. Let's address that now. All the Units in a CogPrime cluster are kept in sync by a global cognitive cycle, whose purpose is described in Section VII.

We mentioned above that each machine in the CogPrime cluster has a housekeeping process. One of its tasks is to keep track of the cognitive cycle, broadcasting when the machine has finished its cycle, and listening to similar broadcasts from its counterparts in the cluster. When all the machines have completed a cycle, a global counter is updated, and each machine is then free to begin the next cycle.

One potential annoyance with this global cognitive cycle is that some machines may complete their cycle much faster than others, and then sit idly while the stragglers finish their jobs. CogPrime addresses this issue in two ways:

- Over the long run, a load balancing process will assign MindAgents from overburdened machines to underutilized ones. The MindAgent migration process is described in the next section.

- In a shorter time horizon, during which a machine's configuration is fixed, there are two heuristics to minimize the waste of processor time without breaking the overall cognitive cycle coordination:
  - The Task queue in each of the machine's Units can be processed more extensively than it would by default; in extreme cases, the machine can go through the whole queue.
  - Background process MindAgents can be given extra activations, as their activity is unlikely to throw the system out of sync, unlike with more focused and goal-oriented processes.

  Both heuristics are implemented by the scheduler inside each unit, which has one boolean trigger for each heuristic. The triggers are set by the housekeeping process when it observes that the machine has been frequently idle over the recent past, and then reset if the situation changes.

19.5 Distributed AtomSpace and Cognitive Dynamics

As hinted above, realistic CogPrime deployments will be spread around reasonably large clusters of co-located machines. This section describes how this distributed deployment scenario is planned for in the design of the AtomSpace and the MindAgents, and how the cognitive dynamics take place in such a scenario.

We won't review the standard principles of distributed computing here, but we will focus on specific issues that arise when the CogPrime is spread across a relatively large number of machines. The two key issues that need to be handled are:

- How to distribute knowledge (i.e., the AtomSpace) in a way that doesn't impose a large performance penalty?
- How to allocate resources (i.e., machines) to the different cognitive processes (MindAgents) in a way that's flexible and dynamic?

19.5.1 Distributing the AtomSpace

The design of a distributed AtomSpace was guided by the following high level requirements:

1. Scale up, transparently, to clusters of dozens to hundreds of machines, without requiring a single central master server.
2. The ability to store portions of an Atom repository on a number of machines in a cluster, where each machine also runs some MindAgens. The distribution of Atoms across the machines should benefit from the fact

that the cognitive processes on one machine are likely to access local Atoms more often than remote ones.
3. Provide transparent access to all Atoms in RAM to all machines in the cluster, even if at different latency and performance levels.
4. For local access to Atoms in the same machine, performance should be as close as possible to what one would have in a similar, but non-distributed AtomSpace.
5. Allow multiple copies of the same Atom to exist in different machines of the cluster, but only one copy per machine.
6. As Atoms are updated fairly often by cognitive dynamics, provide a mechanism for eventual consistency. This mechanism needs not only to propagate changes to the Atoms, but sometimes to reconcile incompatible changes, such as when two cognitive processes update an Atom's TruthValue in opposite ways. Consistency is less important than efficiency, but should be guaranteed eventually.
7. Resolution of inconsistencies should be guided by the importance of the Atoms involved, so the more important ones are more quickly resolved.
8. System configuration can explicitly order the placement of some Atoms to specific machines, and mark a subset of those Atoms as immovable, which should ensure that local copies are always kept.
9. Atom placement across machines, aside from the immovable Atoms, should be dynamic, rebalancing based on frequency of access to the Atom by the different machines.

The first requirement follows obviously from our estimates of how many machines CogPrime will require to display advanced intelligence.

The second requirement above means that we don't have two kinds of machines in the cluster, where some are processing servers and some are database servers. Rather, we prefer each machine to store some knowledge and host some processes acting on that knowledge. This design assumes that there are simple heuristic ways to partition the knowledge across the machines, resulting in allocations that, most of the time, give the MindAgents local access to the Atoms they need most often.

Alas, there will always be some cases in which a MindAgent needs an Atom that isn't available locally. In order to keep the design on the MindAgents simple, this leads to the third requirement, transparency, and to the fourth one, performance.

This partition design, on the other hand, means that there must be some replication of knowledge, as there will always be some Atoms that are needed often by MindAgents on different machines. This leads to requirement five (allow redundant copies of an Atom). However, as MindAgents frequently update the mutable components of Atoms, requirements six and seven are needed, to minimize the impact of conflicts on system performance while striving to guarantee that conflicts are eventually solved, and with priority proportional to the importance of the impacted Atoms.

19.5.1.1 Mechanisms of Managing Distributed Atomspaces

When one digs into the details of distributed AtomSpaces, a number of subtleties emerge. Going into these in full detail here would not be appropriate, but we will make a few comments, just to give a flavor of the sorts of issues involved.

To discuss these issues clearly, some special terminology is useful. In this context, it is useful to reserve the word "Atom" for its pure, theoretical definition, viz: "a Node is uniquely determined by its name. A Link is uniquely determined by its outgoing set". Atoms sitting in RAM may then be called "Realized Atoms". Thus, given a single, pure "abstract/theoretical" Atom, there might be two different Realized Atoms, on two different servers, having the same name/outgoing-set. It's OK to think of a RealizedAtom as a clone of the pure, abstract Atom, and to talk about it that way. Analogously, we might call atoms living on disk, or flying on a wire, "Serialized Atoms"; and, when need be, use specialized terms like "ZMQ-serialized atoms", or "BerkeleyDB-serialized Atoms", etc.

An important and obvious coherency requirement is: "If a MindAgent asks for the Handle of an Atom at time A, and then asks, later on, for the Handle of the same Atom, it should receive the same Handle."

By the "AtomSpace", in general, we mean the container(s) that are used to store the set of Atoms used in an OpenCog system, both in RAM and on disk. In the case of an Atom space that is distributed across multiple machines or other data stores, we will call each of these an "Atom space portion"

Atoms and Handles

Each OpenCog Atom is associated with a Handle object, which is used to identify the Atom uniquely. The Handle is a sort of "key" used, at the infrastructure level, to compactly identify the Atom. In a single-machine, non-distributed Atomspace, one can effectively just use long ints as Handles, and assign successive ints as Handles to successively created new Atoms. In a distributed Atomspace, it's a little subtler. Perhaps the cleanest approach in this case is to use a hash of the serialized Atom data as the handle for an Atom. That way, if an Atom is created in any portion, it will inherently have the same handle as any of its clones.

The issue of Handle collisions then occurs – it is possible, though it will be rare, that two different Atoms will be assigned the same Handle via the hashing function. This situation can be identified via checking, when an Atom is imported into a portion, whether there is already some Atom in that portion with the same Handle but different fundamental aspects. In the rare occasion where this situation does occur, one of the Atoms must then have its Handle changed. Changing an Atom's handle everywhere it's referenced in RAM is not a big deal, so long as it only happens occasionally. However, some sort

of global record of Handle changes should be kept, to avoid confusion in the process of loading saved Atoms from disk. If a loaded Atomspace contains Atoms that have changed Handle since the file was saved, the Atom loading process needs to know about this.

The standard mathematics of hash functions collisions, shows that if one has a space of H possible Handles, one will get two Atoms with the same Handle after $1.25 \times \sqrt{(H)}$ tries, on average.... Rearranging this, it means we'd need a space of around N^2 Handles to have a space of Handles for N possible Atoms, in which one collision would occur on average.... So to have a probability of one collision, for N possible Atoms, one would have to use a handle range up to N^2. The number of bits needed to encode N^2 is twice as many as the number needed to encode N. So, if one wants to minimize collisions, one may need to make Handles twice as long, thus taking up more memory.

However, this memory cost can be palliated via introducing "local Handles" separate from the global, system-wide Handles. The local Handles are used internally within each local Atomspace, and then each local Atomspace contains a translation table going back and forth between local and global Handles. Local handles may be long ints, allocated sequentially to each new Atom entered into a portion. Persistence to disk would always use the global Handles.

To understand the memory tradeoffs involved in these solutions, assume that the global Handles were k times as long as the local handles... and suppose that the average Handle occurred r times in the local Atomspace. Then the memory inflation ratio of the local/global solution as opposed to a solution using only the shorter local handles, would be

$$(1 + k + r)/r = 1 + (k + 1)/r$$

if k=2 and r=10 (each handle is used 10 times on average, which is realistic based on current real-world OpenCog Atomspaces), then the ratio is just $1.3x$ – suggesting that using hash codes for global Handles, and local Handles to save memory in each local AtomSpace, is acceptable memory-wise.

19.5.1.2 Distribution of Atoms

Given the goal of maximizing the probability that an Atom will be local to the machines of the MindAgents that need it, the two big decisions are how to allocate Atoms to machines, and then how to reconcile the results of MindAgents actuating on those Atoms.

The initial allocation of Atoms to machines may be done via explicit system configuration, for Atoms known to have different levels of importance to specific MindAgents. That is, after all, how MindAgents are initially allocated to machines as well.

One may, for instance, create a CogPrime cluster where one machine (or group) focuses on visual perception, one focuses on language processing, one focuses on abstract reasoning, etc. In that case one can hard-wire the location of Atoms.

What if one wants to have three abstract-reasoning machines in one's cluster? Then one can define an abstract-reasoning zone consisting of three Atom repository portions. One can hard-wire that Atoms created by MindAgents in the zone must always remain in that zone – but can potentially be moved among different portions within that zone, as well as replicated across two or all of the machines, if need be. By default they would still initially be placed in the same portion as the MindAgent that created them.

However Atoms are initially placed in portions, sometimes it will be appropriate to move them. And sometimes it will be appropriate to clone an Atom, so there's a copy of it in a different portion from where it exists. Various algorithms could work for this, but the following is one simple mechanism:

- When an Atom A in machine M_1 is requested by a MindAgent in machine M_2, then a clone of A is temporarily created in M_2.
- When an Atom is forgotten (due to low LTI), then a check is made if it has any clones, and any links to it are changed into links to its clones.
- The LTI of an Atom may get a boost if that Atom has no clones (the amount of this boost is a parameter that may be adjusted).

19.5.1.3 MindAgents and the Distributed AtomSpace

In the context of a distributed AtomSpace, the interactions between MindAgents and the knowledge store become subtler, as we'll now discuss.

When a MindAgent wants to create an Atom, it will make this request of the local AtomSpace process, which hosts a subset of the whole AtomSpace. It can, on Atom creation, specify whether the Atom is immovable or not. In the former case, it will initially only be accessible by the MindAgents in the local machine.

The process of assigning the new Atom an AtomHandle needs to be taken care of, in a way that doesn't introduce a central master. One way to achieve that is to make handles hierarchical, so the higher order bits indicate the machine. This, however, means that AtomHandles are no longer immutable. A better idea is to automatically allocate a subset of the AtomHandle universe to each machine. The initial use of those AtomHandles is the privilege of that machine but, as Atoms migrate or are cloned, the handles can move through the cluster.

When a MindAgent wants to retrieve one or more Atoms, it will perform a query on the local AtomSpace subset, just as it would with a single machine repository. Along with the regular query parameters, it may specify whether the request should be processed locally only, or globally. Local queries will be

fast, but may fail to retrieve the desired Atoms, while global queries may take a while to return. In the approach outlined above for MindAgent dynamics and scheduling, this would just cause the MindAgent to wait until results are available.

Queries designed to always return a set of Atoms can have a third mode, which is "prioritize local Atoms". In this case, the AtomSpace, when processing a query that looks for Atoms that match a certain pattern would try to find all local responses before asking other machines.

19.5.1.4 Conflict Resolution

A key design decision when implementing a distributed AtomSpace is the trade-off between consistency and efficiency. There is no universal answer to this conflict, but the usage scenarios for CogPrime , current and planned, tend to fall on the same broad category as far consistency goes. CogPrime 's cognitive processes are relatively indifferent to conflicts and capable of working well with outdated data, especially if the conflicts are temporary. For applications such as the AGI Preschool, it is unlikely that outdated properties of single Atoms will have a large, noticeable impact on the system's behavior; even if that were to happen on rare occasions, this kind of inconsistency if often present in human behavior as well.

On the other hand, CogPrime assumes fairly fast access to Atoms by the cognitive processes, so efficiency shouldn't be too heavily penalized. The robustness against mistakes and the need for performance mean that a distributed AtomSpace should follow the principle of "eventual consistency". This means that conflicts are allowed to arise, and even to persist for a while, but a mechanism is needed to reconcile them.

Before describing conflict resolution, which in CogPrime is a bit more complicated than in most applications, we note that there are two kinds of conflicts. The simple one happens when an Atom that exists in multiple machines is modified in one machine, and that change isn't immediately propagated. The less obvious one happens when some process creates a new Atom in its local AtomSpace repository, but that Atom conceptually "already exists" elsewhere in the system. Both scenarios are handled in the same way, and can become complicated when, instead of a single change or creation, one needs to reconcile multiple operations.

The way to handle conflicts is to have a special purpose control process, a reconciliation MindAgent, with one copy running on each machine in the cluster. This MindAgent keeps track of all recent write operations in that machine (Atom creations or changes).

Each time the reconciliation MindAgent is called, it processes a certain number of Atoms in the recent writes list. It chooses the Atoms to process based on a combination of their STI, LTI and recency of creation/change. Highest priority is given to Atoms with higher STI and LTI that have been

around longer. Lowest priority is given to Atoms with low STI or LTI that have been very recently changed – both because they may change again in the very near future, and because they may be forgotten before it's worth solving any conflicts. This will be the case with most perceptual Atoms, for instance.

By tuning how many Atoms this reconciliation MindAgent processes each time it's activated we can tweak the consistency vs efficiency trade-off.

When the AtomReconciliation agent processes an Atom, what it does is:

- Searches all the machines in the cluster to see if there are other equivalent Atoms (for Nodes, these are Atoms with the same name and type; for Links, these are Atoms with the same type and targets).
- If it finds equivalent Atoms, and there are conflicts to be reconciled, such as different TruthValues or AttentionValues, the decision of how to handle the conflicts is made by a special probabilistic reasoning rule, called the Rule of Choice (see Chapter decides what to do, using the PLN Rule of Choice (see Chapter 34). Basically, this means:
  - It decided whether to merge the conflicting Atoms. We always merge Links, but some Nodes may have different semantics, such as Nodes representing different procedures that have been given the same name.
  - In the case that the two Atoms A and B should be merged, it creates a new Atom C that has all the same immutable properties as A and B. It merges their TruthValues according to the probabilistic revision rule (see Chapter 34). The AttentionValues are merged by prioritizing the higher importances.
  - In the case that two Nodes should be allowed to remain separate, it allocates one of them (say, B) a new name. Optionally, it also evaluates whether a SimilarityLink should be created between the two different Nodes.

Another use for the reconcilitation MindAgent is maintaining approximate consistency between clones, which can be created by the AtomSpace itself, as described above in Subsection 19.5.1.2. When the system knows about the multiple clones of an Atom, it keeps note of these versions in a list, which is processed periodically by a conflict resolution MindAgent, in order to prevent the clones from drifting too far apart by the actions of local cognitive processes in each machine.

19.5.2 Distributed Processing

The OCF infrastructure as described above already contains a lot of distributed processing implicit in it. However, it doesn't tell you how to make

the complex cognitive processes that are part of the CogPrime design distributed unto themselves – say, how to make PLN or MOSES themselves distributed. This turns out to be quite possible, but becomes quite intricate and specific depending on the particular algorithms involved. For instance, the current MOSES implementation is now highly amenable to distributed and multiprocessor implementation, but in a way that depends subtly on the specifics of MOSES and has little to do with the role of MOSES in CogPrime as a whole. So we will not delve into these topics here.

Another possibility worth mentioning is broadly distributed processing, in which CogPrime intelligence is spread across thousands or millions of relatively weak machines networked via the Internet. Even if none of these machines is exclusively devoted to CogPrime , the total processing power may be massive, and massively valuable. The use of this kind of broadly distributed computing resource to help CogPrime is quite possible, but involves numerous additional control problems which we will not address here.

A simple case is massive global distribution of MOSES fitness evaluation. In the case where fitness evaluation is isolated and depends only on local data, this is extremely straightforward. In the more general case where fitness evaluation depends on knowledge stored in a large AtomSpace, it requires a subtler design, wherein each globally distributed MOSES subpopulation contains a pool of largely similar genotypes, and contains a cache of relevant parts of the AtomSpace, which is continually refreshed during the fitness evaluation process. This can work so long as each globally distributed lobe has a reasonably reliable high-bandwidth connection to a machine containing a large AtomSpace.

On the more mundane topic of distributed processing within the main CogPrime cluster, three points are worth discussing:

- Distributed communication and coordination between MindAgents.
- Allocation of machines to functional groups, and MindAgent migration.
- Machines entering and leaving the cluster.

19.5.2.1 Distributed Communication and Coordination

Communications between MindAgents, Units and other CogPrime components are handled by a message queue subsystem. This subsystem provides a unified API, so the agents involved are unaware of the location of their partners: distributed messages, inter-process messages in the same machine, and intra-process messages in the same Unit are sent through the same API, and delivered to the same target queues. This design enables transparent distribution of MindAgents and other components.

In the simplest case, of MindAgents within the same Unit, messages are delivered almost immediately, and will be available for processing by the target agent the next time it's enacted by the scheduler. In the case of messages

sent to other Units or other machines, they're delivered to the messaging subsystem component of that unit, which has a dedicated thread for message delivery. That subsystem is scheduled for processing just like any other control process, although it tends to have a reasonably high importance, to ensure speedy delivery.

The same messaging API and subsystem is used for control-level communications, such as the coordination of the global cognitive cycle. The cognitive cycle completion message can be used for other housekeeping contents as well.

19.5.2.2 Functional Groups and MindAgent Migration

A CogPrime cluster is composed of groups of machines dedicated to various high-level cognitive tasks: perception processing, language processing, background reasoning, procedure learning, action selection and execution, goal achievement planning, etc. Each of these high-level tasks will probably require a number of machines, which we call functional groups.

Most of the support needed for functional groups is provided transparently by the mechanisms for distributing the AtomSpace and by the communications layer. The main issue is how much resources (i.e., how many machines) to allocate to each functional group. The initial allocation is determined by human administrators via the system configuration – each machine in the cluster has a local configuration file which tells it exactly which processes to start, along with the collection of MindAgents to be loaded onto each process and their initial AttentionValues.

Over time, however, it may be necessary to modify this allocation, adding machines to overworked or highly important functional groups. For instance, one may add more machines to the natural language and perception processing groups during periods of heavy interaction with humans in the preschool environment, while repurposing those machines to procedure learning and background inference during periods in which the agent controlled by CogPrime is resting or "sleeping".

This allocation of machines is driven by attention allocation in much the same way that processor time is allocated to MindAgents. Functional groups can be represented by Atoms, and their importance levels are updated according to the importance of the system's top level goals, and the usefulness of each functional group to their achievement. Thus, once the agent is engaged by humans, the goals of pleasing them and better understanding them would become highly important, and would thus drive the STI of the language understanding and language generation functional groups.

Once there is an imbalance between a functional group's STI and its share of the machines in the cluster, a control process CIM-Dynamic is triggered to decide how to reconfigure the cluster. This CIM-Dynamic works approximately as follows:

- First, it decides how many extra machines to allocate to each sub-represented functional group.
- Then, it ranks the machines not already allocated to those groups based on a combination of their workload and the aggregate STI of their MindAgents and Units. The goal is to identify machines that are both relatively unimportant and working under capacity.
- It will then migrate the MindAgents of those machines to other machines in the same functional group (or just remove them if clones exist), freeing them up.
- Finally, it will decide how best to allocate the new machines to each functional group. This decision is heavily dependent on the nature of the work done by the MindAgents in that group, so in CogPrime these decisions will be somewhat hardcoded, as is the set of functional groups. For instance, background reasoning can be scaled just by adding extra inference MindAgents to the new machines without too much trouble, but communicating with humans requires MindAgents responsible for dialog management, and it doesn't make sense to clone those, so it's better to just give more resources to each MindAgent without increasing their numbers.

The migration of MindAgents becomes, indirectly, a key driver of Atom migration. As MindAgents move or are cloned to new machines, the AtomSpace repository in the source machine should send clones of the Atoms most recently used by these MindAgents to the target machine(s), anticipating a very likely distributed request that would create those clones in the near future anyway. If the MindAgents are moved but not cloned, the local copies of those Atoms in the source machine can then be (locally) forgotten.

19.5.2.3 Adding and Removing Machines

Given the support for MindAgent migration and cloning outlined above, the issue of adding new machines to the cluster becomes a specific application of the heuristics just described. When a new machine is added to the cluster, CogPrime initially decides on a functional group for it, based both on the importance of each functional group and on their current performance – if a functional group consistently delays the completion of the cognitive cycle, it should get more machines, for instance. When the machine is added to a functional group, it is then populated with the most important or resource starved MindAgents in that group, a decision that is taken by economic attention allocation.

Removal of a machine follows a similar process. First the system checks if the machine can be safely removed from its current functional group, without greatly impacting its performance. If that's the case, the non-cloned MindAgents in that machine are distributed among the remaining machines in the group, following the heuristic described above for migration. Any local-only

Atoms in that machine's AtomSpace container are migrated as well, provided their LTI is high enough.

In the situation in which removing a machine M_1 would have an intolerable impact on the functional group's performance, a control process selects another functional group to lose a machine M_2. Then, the MindAgents and Atoms in M_1 are migrated to M_2, which goes through the regular removal process first.

In principle, one might use the insertion or removal of machines to perform a global optimization of resource allocation within the system, but that process tends to be much more expensive than the simpler heuristics we just described. We believe these heuristics can give us most of the benefits of global re-allocation at a fraction of the disturbance for the system's overall dynamics during their execution.

Chapter 20
Knowledge Representation Using the Atomspace

20.1 Introduction

CogPrime 's knowledge representation must be considered on two levels: implicit and explicit. This chapter considers mainly explicit knowledge representation, with a focus on representation of declarative knowledge. We will describe the Atom knowledge representation, a generalized hypergraph formalism which comprises a specific vocabulary of Node and Link types, used to represent declarative knowledge but also, to a lesser extent, other types of knowledge as well. Other mechanisms of representing procedural, episodic, attentional, and intentional knowledge will be handled in later chapters, as will the subtleties of implicit knowledge representation.

The AtomSpace Node and Link formalism is the most obviously distinctive aspect of the OpenCog architecture, from the point of view of a software developer building AI processes in the OpenCog framework. But yet, the features of CogPrime that are most important, in terms of our theoretical reasons for estimating it likely to succeed as an advanced AGI system, are not really dependent on the particulars of the AtomSpace representation.

What's important about the AtomSpace knowledge representation is mainly that it provides a flexible means for compactly representing multiple forms of knowledge, in a way that allows them to interoperate – where by "interoperate" we that e.g. a fragment of a chunk of declarative knowledge can link to a fragment of a chunk of attentional or procedural knowledge; or a chunk of knowledge in one category can overlap with a chunk of knowledge in another category (as when the same link has both a (declarative) truth value and an (attentional) importance value). In short, any representational infrastructure sufficiently flexible to support

- compact representation of all the key categories of knowledge playing dominant roles in human memory

- the flexible creation of specialized sub-representations for various particular subtypes of knowledge in all these categories, enabling compact and rapidly manipulable expression of knowledge of these subtypes
- the overlap and interlinkage of knowledge of various types, including that represented using specialized sub-representations

will probably be acceptable for CogPrime 's purposes. However, precisely formulating these general requirements is tricky, and is significantly more difficult than simply articulating a single acceptable representational scheme, like the current OpenCog Atom formalism. The Atom formalism satisfies the relevant general requirements and has proved workable from a practical software perspective.

In terms of the Mind-World Correspondence Principle introduced in Chapter 10, the important point regarding the Atom representation is that it must be flexible enough to allow the compact and rapidly manipulable representation of knowledge that has aspects spanning the multiple common human knowledge categories, in a manner that allows easy implementation of cognitive processes that will manifest the Mind-World Correspondence Principle in everyday human-like situations. The actual manifestation of mind-world correspondence is the job of the cognitive processes acting on the AtomSpace – the job of the AtomSpace is to be an efficient and flexible enough representation that these cognitive processes can manifest mind-world correspondence in everyday human contexts given highly limited computational resources.

20.2 Denoting Atoms

First we describe the textual notation we'll use to denote various sorts of Atom throughout the following chapters. The discussion will also serve to give some particular examples of cognitively meaningful Atom constructs.

20.2.1 Meta-Language

As always occurs when discussing (even partially) logic-based systems, when discussing CogPrime there is some potential for confusion between logical relationships inside the system, and logical relationships being used to describe parts of the system. For instance, we can state as observers that two Atoms inside CogPrime are equivalent, and this is different from stating that CogPrime itself contains an Equivalence relation between these two Atoms. Our formal notation needs to reflect this difference.

Since we will not be doing any fancy mathematical analyses of CogPrime structures or dynamics here, there is no need to formally specify the logic being used for the metalanguage. Standard predicate logic may be assumed.

So, for example, we will say things like

```
(IntensionalInheritanceLink Ben monster).TruthValue.strength = .5
```

This is a metalanguage statement, which means that the strength field of the TruthValue object associated with the link (IntensionalInheritance Ben monster) is equal to .5. This is different than saying

```
EquivalenceLink
    ExOutLink
        GetStrength
        ExOutLink
            GetTruthValue
            IntensionalInheritanceLink Ben monster
    NumberNode 0.5
```

which refers to an equivalence relation represented inside CogPrime . The former refers to an equals relationship observed by the authors of the book, but perhaps never represented explicitly inside CogPrime .

In the first example above we have used the C++ convention

```
structure_variable_name.field_name
```

for denoting elements of composite structures; this convention will be stated formally below.

In the second example we have used schema corresponding to TruthValue and Strength; these schema extract the appropriate fields from the Atoms they're applied to, so that e.g.

```
ExOutLink
    GetTruthValue
    A
```

returns the number

```
A.TruthValue
```

Following a convention from mathematical logic, we will also sometimes use the special symbol

```
|-
```

to mean "implies in the metalanguage." For example, the first-order PLN deductive inference strength rule may be written

```
InheritanceLink A B <sAB>
InheritanceLink B C <sBC>
|-
InheritanceLink A C <sAC>
```

where

```
sAC = sAB sBC + (1-sAB) ( sC - sB sBC ) / (1- sB )
```

This is different from saying

```
ForAll $A, $B, $C, $sAB, $sBC, $sAC

ExtensionalImplicationLink_HOJ
    AND
        InheritanceLink $A $B <$sAB>
        InheritanceLink $B $C <$sBC>
    AND
        InheritanceLink $A $C <$sAC>
        $sAC = $sAB $sBC + (1-$sAB) ($sC - $sB $sBC) / (1- $sB)
```

which is the most natural representation of the independence-based PLN deduction rule (for strength-only truth values) as a logical statement within CogPrime . In the latter expression the variables $A, $sAB, and so forth represent actual Variable Atoms within CogPrime . In the former expression the variables represent concrete, non-Variable Atoms within CogPrime , which however are being considered as variables within the metalanguage.

(As explained in the PLN book, a link labeled with "HOJ" refers to a "higher order judgment", meaning a relationship that interprets its relations as entities with particular truth values. For instance,

```
ImplicationLink_HOJ
   Inh $X stupid <.9>
   Inh $X rich <.9>
```

means that if (Inh $X stupid) has a strength of .9, then (Inh $X rich) has a strength of .9). WIKISOURCE:AtomNotation

20.2.2 Denoting Atoms

Atoms are the basic objects making up CogPrime knowledge. They come in various types, and are associated with various dynamics, which are embodied in MindAgents. Generally speaking Atoms are endowed with TruthValue and AttentionValue objects. They also sometimes have names, and other associated Values as previously discussed. In the following subsections we will explain how these are notated, and then discuss specific notations for Links and Nodes, the two types of Atoms in the system.

20.2.2.1 Names

In order to denote an Atom in discussion, we have to call it something. Relatedly but separately, Atoms may also have names within the CogPrime system. (As a matter of implementation, in the current OpenCog version, no Links have names; whereas, all Nodes have names, but some Nodes have a null name, which is conceptually the same as not having a name.)

(name,type) pairs must be considered as unique within each Unit within a OpenCog system, otherwise they can't be used effectively to reference Atoms.

It's OK if two different OpenCog Units both have SchemaNodes named "+", but not if one OpenCog Unit has two SchemaNodes both named "+" - this latter situation is disallowed on the software level, and is assumed in discussions not to occur.

Some Atoms have natural names. For instance, the SchemaNode corresponding to the elementary schema function + may quite naturally be named "+". The NumberNode corresponding to the number .5 may naturally be named ".5", and the CharacterNode corresponding to the character *c* may naturally be named "c". These cases are the minority, however. For instance, a SpecificEntityNode representing a particular instance of + has no natural name, nor does a SpecificEntityNode representing a particular instance of c.

Names should not be confused with Handles. Atoms have Handles, which are unique identifiers (in practice, numbers) assigned to them by the OpenCog core system; and these Handles are how Atoms are referenced internally, within OpenCog, nearly all the time. Accessing of Atoms by name is a special case - not all Atoms have names, but all Atoms have Handles. An example of accessing an Atom by name is looking up the CharacterNode representing the letter "c" by its name "c". There would then be two possible representations for the word "cat":

1. this word might be associated with a ListLink - and the ListLink corresponding to "cat" would be a list of the Handles of the Atoms of the nodes named "c", "a", and "t".
2. for expedience, the word might be associated with a WordNode named "cat."

In the case where an Atom has multiple versions, this may happen for instance if the Atom is considered in a different context (via a ContextLink), each version has a VersionHandle, so that accessing an AtomVersion requires specifying an AtomHandle plus a VersionHandle. See Chapter 19 for more information on Handles.

OpenCog never assigns Atoms names *on its own*; in fact, Atom names are assigned only in the two sorts of cases just mentioned:

1. Via preprocessing of perceptual inputs (e.g. the names of NumberNode, CharacterNodes)
2. Via hard-wiring of names for SchemaNodes and PredicateNodes corresponding to built-in elementary schema (e.g. +, AND, Say)

If an Atom A has a name n in the system, we may write

```
A.name = n
```

On the other hand, if we want to assign an Atom an *external* name, we may make a meta-language assertion such as

```
L1 := (InheritanceLink Ben animal)
```

indicating that we decided to name that link L1 for our discussions, even though inside OpenCog it has no name.

In denoting (nameless) Atoms we may use arbitrary names like L1. This is more convenient than using a Handle based notation which Atoms would be referred to as 1, 3433322, etc.; but sometimes we will use the Handle notation as well.

Some ConceptNodes and conceptual PredicateNode or SchemaNodes may correspond with human-language words or phrases like *cat*, *bite*, and so forth. This will be the minority case; more such nodes will correspond to parts of human-language concepts or fuzzy collections of human-language concepts. In discussions in this book, however, we will often invoke the unusual case in which Atoms correspond to individual human-language concepts. This is because such examples are the easiest ones to write about and discuss intuitively. The preponderance of named Atoms in the examples in the book implies no similar preponderance of named Atoms in the real OpenCog system. It is merely easier to talk about a hypothetical Atom named "cat" than it is about a hypothetical Atom with Handle 434. It is not impossible that a OpenCog system represents "cat" as a single ConceptNode, but it is just as likely that it will represent "cat" as a map composed of many different nodes without any of these having natural names. Each OpenCog works out for itself, implicitly, which concepts to represent as single Atoms and which in distributed fashion.

For another example,

```
ListLink
   CharacterNode ``c''
   CharacterNode ``a''
   CharacterNode ``t''
```

corresponds to the character string

```
(``c'', ``a'', ``t'')
```

and would naturally be named using the string *cat*. In the system itself, however, this ListLink need not have any name.

20.2.2.2 Types

Atoms also have types. When it is necessary to explicitly indicate the type of an atom, we will use the keyword Type, as in

```
A.Type = InheritanceLink

N_345.Type = ConceptNode
```

On the other hand, there is also a built-in schema HasType which lets us say

```
EvaluationLink HasType A InheritanceLink
```

```
EvaluationLink HasType N_345 ConceptNode
```

This covers the case in which type evaluation occurs explicitly in the system, which is useful if the system is analyzing its own emergent structures and dynamics.

Another option currently implemented in OpenCog is to explicitly restrict the type of a variable using `TypedVariableLink` such as follows

```
TypedVariableLink
    VariableNode $X
    VariableTypeNode "ConceptNode"
```

Note also that we will frequently remove the suffix `Link` or `Node` from their type name, such as

```
Inheritance
    Concept A
    Concept B
```

instead of

```
InheritanceLink
    ConceptNode A
    ConceptNode B
```

20.2.2.3 Truth Values

The truth value of an atom is a bundle of information describing how *true* the Atom is, in one of several different senses depending on the Atom type. It is encased in a TruthValue object associated with the Atom. Most of the time, we will denote the truth value of an atom in <>'s following the expression denoting the atom. This very handy notation may be used in several different ways.

A complication is that some Atoms may have CompositeTruthValues, which consist of different estimates of their truth value made by different sources, which for whatever reason have not been reconciled (maybe no process has gotten around to reconciling them, maybe they correspond to different truth values in different contexts and thus logically need to remain separate, maybe their reconciliation is being delayed pending accumulation of more evidence, etc.). In this case we can still assume that an Atom has a default truth value, which corresponds to the highest-confidence truth value that it has, in the Universal Context.

Most frequently, the notation is used with a single number in the brackets, e.g.

```
A <.4>
```

to indicate that the atom A has truth value .4; or

```
IntensionalInheritanceLink Ben monster <.5>
```

to indicate that the IntensionalInheritance relation between Ben and monster has truth value strength .5. In this case, <tv> indicates (roughly speaking) that the truth value of the atom in question involves a probability distribution with a mean of tv. The precise semantics of the strength values associated with OpenCog Atoms is described in Probabilistic Logic Networks (see Chapter 34). Please note, though: This notation does not imply that the only data retained in the system about the distribution is the single number .5.

If we want to refer to the truth value of an Atom A in the context C, we can use the construct

```
ContextLink <truth value>
  C
  A
```

Sometimes, Atoms in OpenCog are labeled with two truth value components as defined by PLN: strength and weight-of-evidence. To denote these two components, we might write

```
IntensionalInheritanceLink Ben scary <.9,.1>
```

indicating that there is a relatively small amount of evidence in favor of the proposition that Ben is very scary.

We may also put the TruthValue indicator in a different place, e.g. using indent notation,

```
IntensionalInheritanceLink <.9,.1>
    Ben
    scary
```

This is mostly useful when dealing with long and complicated constructions.

If we want to denote a composite truth value (whose components correspond to different "versions" of the Atom), we can use a list notation, e.g.

```
IntensionalInheritance (<.9,.1>, <.5,.9> [h,123],<.6,.7> [c,655])
    Ben
    scary
```

where e.g.

```
<.5,.9> [h,123]
```

denotes the TruthValue version of the Atom indexed by Handle 123. The `h` denotes that the AtomVersion indicated by the VersionHandle h,123 is a Hypothetical Atom, in the sense described in the PLN book. Some versions may not have any index Handles.

The semantics of composite TruthValues are described in the PLN book, but roughly they are as follows. Any version not indexed by a VersionHandle is a "primary TruthValue" that gives the truth value of the Atom based on some body of evidence. A version indexed by a VersionHandle is either contextual or hypothetical, as indicated notationally by the c or h in its VersionHandle. So, for instance, if a TruthValue version for Atom A has VersionHandle h,123 that means it denotes the truth value of Atom A under the hypothetical

context represented by the Atom with handle 123. If a TruthValue version for Atom A has VersionHandle c,655 this means it denotes the truth value of Atom A in the context represented by the Atom with Handle 655.

Alternately, truth values may be expressed sometimes in <L,U,b> or <L,U,b,N> format, defined in terms of indefinite probability theory as defined in the PLN book and recalled in Chapter 34. For instance,

```
IntensionalInheritanceLink Ben scary <.7,.9,.8,20>
```

has the semantics that *There is an estimated 80% chance that after 20 more observations have been made, the estimated strength of the link will be in the interval (.7,.9).*

The notation may also be used to specify a TruthValue probability distribution, e.g.

```
A <g(5,7,12)>
```

would indicate that the truth value of A is given by distribution g with parameters (5,7,12), or

```
A <M>
```

where M is a table of numbers, would indicate that the truth value of A is approximated by the table M.

The <> notation for truth value is an unabashedly incomplete and ambiguous notation, but it is very convenient. If we want to specify, say, that the truth value strength of IntensionalInheritanceLink Ben monster is in fact the number .5, and no other truth value information is retained in the system, then we need to say

```
(IntensionalInheritance Ben monster).TruthValue = [(strength, .5)]
```

(where a hashtable form is assumed for TruthValue objects, i.e. a list of name-value pairs). But this kind of issue will rarely arise here and the <> notation will serve us well.

20.2.2.4 Attention Values

The AttentionValue object associated with an Atom does not need to be notated nearly as often as truth value. When it does however we can use similar notational methods.

AttentionValues may have several components, but the two critical ones are called short-term importance (STI) and long-term importance (LTI). Furthermore, multiple STI values are retained: for each (Atom, MindAgent) pair there may be a Mind-Agent-specific STI value for that Atom. The pragmatic import of these values will become clear in a later chapter when we discuss attention allocation.

Roughly speaking, the long-term importance is used to control memory usage: when memory gets scarce, the atoms with the lowest LTI value are

removed. On the other hand, the short-term importance is used to control processor time allocation: MindAgents, when they decide which Atoms to act on, will generally, but not only, choose the ones that have proved most useful to them in the recent past, and additionally those that have been useful for other MindAgents in the recent past.

We will use the double bracket <<>> to denote attention value (in the rare cases where such denotation is necessary). So, for instance,

```
Cow_7 <<.5>>
```

will mean the node Cow_7 has an importance of .5; whereas,

```
Cow_7 <<STI=.1, LTI = .8>>
```

or simply

```
Cow_7 <<.1, .8>>
```

will mean the node Cow_7 has short-term importance = .1 and long-term importance = .8 .

Of course, we can also use the style

```
(IntensionalInheritanceLink Ben monster).AttentionValue
                                    = [(STI,.1), (LTI, .8)]
```

where appropriate.

20.2.2.5 Links

Links are represented using a simple notation that has already occurred many times in this book. For instance,

```
Inheritance A B
```

```
Similarity A B
```

Note that here the symmetry or otherwise of the link is not implicit in the notation. SimilarityLinks are symmetrical, InheritanceLinks are not. When this distinction is necessary, it will be explicitly made. WIKISOURCE:FunctionNotation

20.3 Representing Functions and Predicates

SchemaNodes and PredicateNodes contain functions internally; and Links may also usefully be considered as functions. We now briefly discuss the representations and notations we will use to indicate functions in various contexts.

Firstly, we will make some use of the currying notation drawn from combinatory logic, in which adjacency indicates function application. So, for instance, using currying,

```
f x
```

means the function f evaluated at the argument x; and (f x y) means (f(x))(y). If we want to specify explicitly that a block of terminology is being specified using currying we will use the notation @[expression], for instance

```
@[f x y z]
```

means

```
((f(x))(y))(z)
```

We will also frequently use conventional notation to refer to functions, such as f(x,y). Of course, this is consistent with the currying convention if (x,y) is interpreted as a list and f is then a function that acts on 2-element lists. We will have many other occasions than this to use list notation.

Also, we will sometimes use a non-curried notation, most commonly with Links, so that e.g.

```
InheritanceLink x y
```

does not mean a curried evaluation but rather means InheritanceLink(x,y).

20.3.0.6 Execution Output Links

In the case where f refers to a schema, the occurrence of the combination f x in the system is represented by

```
ExOutLink f x
```

or graphically

```
      @
    /   \
   f     x
```

Note that, just as when we write

```
f (g x)
```

we mean to apply f to the result of applying g to x, similarly when we write

```
ExOutLink f (ExOutLink g x)
```

we mean the same thing. So for instance

```
EvaluationLink (ExOutLink g x) y <.8>
```

means that the result of applying g to x is a predicate r, so that r(y) evaluates to True with strength .8.

This approach, in its purest incarnation, does not allow multi-argument schemata. Now, multi-argument schemata are never actually necessary, because one can use argument currying to simulate multiple arguments. However, this is often awkward, and things become simpler if one introduces an explicit tupling operator, which we call ListLink. Simply enough,

```
ListLink A1 ... An
```

denotes an ordered list (A1, . . . , An)

20.3.1 Execution Links

ExecutionLinks give the system an easy way to record acts of schema execution. These are ternary links of the form:

```
SchemaNode: S

Atom: A, B

ExecutionLink S A B
```

In words, this says the procedure represented by SchemaNode S has taken input A and produced output B.

There may also be schemata that do not take output, or do not take input. But these are treated as PredicateNodes, to be discussed below; their activity is recorded by EvaluationLinks, not ExecutionLinks.

The TruthValue of an ExecutionLink records how frequently the result encoded in the ExecutionLink occurs. Specifically,

- the TruthValue of (ExecutionLink S A B) tells you the probability of getting B as output, given that you have run schema S on input A
- the TruthValue of (ExecutionLink S A) tells you the probability that if S is run, it is run on input A

Often it is useful to record the time at which a given act of schema execution was carried out; in that case one uses the atTime link, writing e.g.

```
atTimeLink
    T
    ExecutionLink S A B
```

where T is a TimeNode, or else one uses an implicit method such as storing the time-stamp of the ExecutionLink in a core-level data-structure called the TimeServer. The implicit method is logically equivalent to explicitly using atTime, and is treated the same way by PLN inference, but provides significant advantages in terms of memory usage and lookup speed.

For purposes of logically reasoning about schema, it is useful to create binary links representing ExecutionLinks with some of their arguments fixed. We name these as follows:

```
ExecutionLink1 A B means: X so that ExecutionLink X A B

ExecutionLink2 A B means: X so that ExecutionLink A X B

ExecutionLink3 A B means: X so that ExecutionLink A B X
```

Finally, a SchemaNode may be associated with a structure called a Graph. Where S is a SchemaNode,

```
Graph(S) = { (x,y): ExecutionLink S x y }
```

Sometimes, the graph of a SchemaNode may be explicitly embodied as a ConceptNode; other times, it may be constructed implicitly by a MindAgent in analyzing the SchemaNode (e.g. the inference MindAgent).

Note that the set of ExecutionLinks describing a SchemaNode may not define that SchemaNode exactly, because some of them may be derived by inference. This means that the model of a SchemaNode contained in its ExecutionLinks may not actually be a mathematical function, in the sense of assigning only one output to each input. One may have

```
ExecutionLink S X A <.5>

ExecutionLink S X B <.5>
```

meaning that the system does not know whether S(X) evaluates to A or to B. So the set of ExecutionLinks modeling a SchemaNode may constitute a non-function relation, even if the schema inside the SchemaNode is a function.

Finally, what of the case where f x represents the action of a built-in system function f on an argument x? This is an awkward case that would not be necessary if the CogPrime system were revised so that all cognitive functions were carried out using SchemaNodes. However, in the current CogPrime version, where most cognitive functions are carried out using C++ MindAgent objects, if we want CogPrime to study its own cognitive behavior in a statistical way, we need BuiltInSchemaNodes that refer to MindAgents rather than to ComboTrees (or else, we need to represent MindAgents using ComboTrees, which will become practicable once we have a sufficiently efficient Combo interpreter). The semantics here is thus basically the same as where f refers to a schema. For instance we might have

```
ExecutionLink FirstOrderInferenceMindAgent (L1, L2) L3
```

where L1, L2 and L3 are links related by

```
L1
L2
|-
L3
```

according to the first-order PLN deduction rules.

20.3.1.1 Predicates

Predicates are related but not identical to schema, both conceptually and notationally. PredicateNodes involve *predicate schema* which output TruthValue objects. But there is a difference between a SchemaNode embodying a predicate schema and a PredicateNode, which is that a PredicateNode doesn't output a TruthValue, it adjusts its own TruthValue as a result of the output of its own internal predicate schema.

The record of the activity of a PredicateNode is given not by an ExecutionLink but rather by an:

```
EvaluationLink P A <tv>
```

where P is a PredicateNode, A is its input, and <tv> is the truth value assumed by the EvaluationLink corresponding to the PredicateNode being fed the input A. There is also the variant

```
EvaluationLink P <tv>
```

for the case where the PredicateNode P embodies a schema that takes no inputs[1].

A simple example of a PredicateNode is the predicate GreaterThan. In this case we have, for instance

```
EvaluationLink GreaterThan 5 6 <0>

EvaluationLink GreaterThan 5 3 <1>
```

and we also have:

```
EquivalenceLink
    GreaterThan
    ExOutLink
        And
        ListLink
            ExOutLink
                Not
                LessThan
            ExOutLink
                Not
                EqualTo
```

Note how the variables have been stripped out of the expression, see the PLN book for more explanation about that. We will also encounter many commonsense-semantics predicates such as isMale, with e.g.

```
EvaluationLink isMale Ben_Goertzel <1>
```

Schemata that return no outputs are treated as predicates, and handled using EvaluationLinks. The truth value of such a predicate, as a default, is considered as True if execution is successful, and False otherwise.

[1] actually, if P does take some inputs, `EvaluationLink P <tv>` is defined too and tv corresponds to the average of P(X) over all inputs X, this is explained in more depth in the PLN book.

And, analogously to the Graph operator for SchemaNodes, we have for PredicateNodes the SatisfyingSet operator, defined so that the SatisfyingSet of a predicate is the set whose members are the elements that satisfy the predicate. Formally, that is:

```
S = SatisfyingSet P
```

means

```
TruthValue(MemberLink X S)
```

equals

```
TruthValue(EvaluationLink P X)
```

This operator allows the system to carry out advanced logical operations like higher-order inference and unification.

20.3.2 Denoting Schema and Predicate Variables

CogPrime sometimes uses variables to represent the expressions inside schemata and predicates, and sometimes uses variable-free, combinatory-logic-based representations. There are two sorts of variables in the system, either of which may exist either inside compound schema or predicates, or else in the AtomSpace as VariableNodes:

It is important to distinguish between two sorts of variables that may exist in CogPrime :

- Variable Atoms, which may be quantified (bound to existential or universal quantifiers) or unquantified
- Variables that are used solely as function-arguments or local variables inside the "Combo tree" structures used inside some ProcedureNodes (PredicateNodes or SchemaNodes) (to be described below), but are not related to Variable Atoms

Examples of quantified variables represented by Variable Atoms are $X and $Y in:

```
ForAll $X <.0001>
    ExtensionalImplicationLink
        ExtensionalInheritanceLink $X human
        ThereExists $Y
            AND
                ExtensionalInheritanceLink $Y human
                EvaluationLink parent_of ($X, $Y)
```

An example of an unquantified Variable Atom is $X in

```
ExtensionalImplicationLink <.3>
    ExtensionalInheritanceLink $X human
    ThereExists $Y
```

```
        AND
            ExtensionalInheritanceLink $Y human
            EvaluationLink parent_of ($X, $Y)
```

This ImplicationLink says that 30% of humans are parents: a more useful statement than the ForAll Link given above, which says that it is very very unlikely to be true that all humans are parents.

We may also say, for instance,

```
SatisfyingSet( EvaluationLink eats (cat, $X) )
```

to refer to the set of X so that eats(cat, X).

On the other hand, suppose we have the implication

```
Implication
    Evaluation f $X
    Evaluation
        f
        ExOut reverse $X
```

where f is a PredicateNode embodying a mathematical operator acting on pairs of NumberNodes, and reverse is an operator that reverses a list. So, this implication says that the f predicate is commutative. Now, suppose that f is grounded by the formula

```
f(a,b) = (a > b - 1)
```

embodied in a Combo Tree object (which is not commutative but that is not the point), stored in the ProcedureRepository and linked to the PredicateNode for f. These f-internal variables, which I have written here using the letters a and b, are not VariableNodes in the CogPrime AtomTable. The notation we use for these within the textual Combo language, that goes with the Combo Tree formalism, is to replace a and b in this example with #1 and #2, so the above grounding would be denoted

```
f -> (#1 > #2 - 1)
```

version, it is assumed that type restrictions are always crisp, not probabilistically truth-valued. This assumption may be revisited in a later version of the system.

20.3.2.1 Links as Predicates

It is conceptually important to recognize that CogPrime link types may be interpreted as predicates. For instance, when one says

```
InheritanceLink cat animal <.8>
```

indicating an Inheritance relation between cat and animal with a strength .8, effectively one is declaring that one has a predicate giving an output of .8. Depending on the interpretation of InheritanceLink as a predicate, one has either the predicate

```
InheritanceLink cat $X
```

acting on the input

```
animal
```

or the predicate

```
InheritanceLink $X animal
```

acting on the input

```
cat
```

or the predicate

```
InheritanceLink $X $Y
```

acting on the list input

```
(cat, animal)
```

This means that, if we wanted to, we could do away with all Link types except OrderedLink and UnorderedLink, and represent all other Link types as PredicateNodes embodying appropriate predicate schema.

This is not the approach taken in the current codebase. However, the situation is somewhat similar to that with CIM-Dynamics:

- In future we will likely create a revision of CogPrime that regularly revises its own vocabulary of Link types, in which case an explicit representation of link types as predicate schema will be appropriate.
- In the shorter term, it can be useful to treat link types as *virtual predicates,* meaning that one lets the system create SchemaNodes corresponding to them, and hence do some *meta level reasoning* about its own link types.

20.3.3 Variable and Combinator Notation

One of the most important aspects of combinatory logic, from a CogPrime perspective, is that it allows one to represent arbitrarily complex procedures and patterns without using variables in any direct sense. In CogPrime , variables are optional, and the choice of whether or how to use them may be made (by CogPrime itself) on a contextual basis.

This page deals with the representation of *variable expressions* in a variable-free way, in a CogPrime context. The general theory underlying this is well-known, and is usually expressed in terms of the elimination of variables from lambda calculus expressions (*lambda lifting*). Here we will not present this theory but will restrict ourselves to presenting a simple, hopefully illustrative example, and then discussing some conceptual implications.

20.3.3.1 Why Eliminating Variables is So Useful

Before launching into the specifics, a few words about the general utility of variable-free expression may be worthwhile.

Some expressions look simpler to the trained human eye with variables, and some look simpler without them. However, the main reason why eliminating all variables from an expression is sometimes very useful, is that there are automated program-manipulation techniques that work much more nicely on programs (schemata, in CogPrime lingo) without any variables in them.

As will be discussed later (e.g. in the chapter on evolutionary learning, although the same process is also useful for supporting probabilistic reasoning on procedures), in order to mine patterns among multiple schema that all try to do the same (or related) things, we want to put schema into a kind of "hierarchical normal form." The normal form we wish to use generalizes Holman's Elegant Normal Form (which is discussed in Moshe Looks' PhD thesis) to program trees rather than just Boolean trees.

But, putting computer programs into a useful, nicely-hierarchically-structured normal form is a hard problem - it requires one have a pretty nice and comprehensive set of *program transformations*.

But the only general, robust, systematic program transformation methods that exist in the computer science literature require one to remove the variables from one's programs, so that one can use the theory of functional programming (which ties in with the theory of monads in category theory, and a lot of beautiful related math).

So: In large part, we want to remove variables so we can use functional programming tools to normalize programs into a standard and pretty hierarchical form, so we can mine patterns among them effectively.

However, we don't *always* want to be rid of variables, because sometimes, from a logical reasoning perspective, theorem-proving is easier with the variables in there. (Sometimes not.)

So, we want to have the option to use variables, or not.

20.3.3.2 An Example of Variable Elimination

Consider the PredicateNode

```
AND
  InheritanceLink X cat
  eats X mice
```

Here we have used a *syntactically sugared* representation involving the variable X. How can we get rid of the X?

Recall the C combinator (from combinatory logic), defined by

```
C f x y = f y x
```

Using this tool,

```
InheritanceLink X cat
```

becomes

```
C InheritanceLink cat X
```

and

```
eats X mice
```

becomes

```
C eats mice X
```

so that overall we have

```
AND
  C InheritanceLink cat
  C eats mice
```

where the C combinators essentially give instructions as to where the *virtual argument* X should go.

In this case the variable-free representation is basically just as simple as the variable-based representation, so there is nothing to lose and a lot to gain by getting rid of the variables. This won't always be the case - sometimes execution efficiency will be significantly enhanced by use of variables.

WIKISOURCE:TypeInheritance

20.3.4 Inheritance Between Higher-Order Types

Next, this section deals with the somewhat subtle matter of Inheritance between higher-order types. This is needed, for example, when one wants to cross over or mutate two complex schemata, in an evolutionary learning context. One encounters questions like: When mutation replaces a schema that takes integer input, can it replace it with one that takes general numerical input? How about vice versa? These questions get more complex when the inputs and outputs of schema may themselves be schema with complex higher-order types. However, they can be dealt with elegantly using some basic mathematical rules.

Denote the type of a mapping from type T to type S, as `T -> S`. Use the shorthand *inh* to mean *inherits from.* Then the basic rule we use is that

```
T1 -> S1 inh T2 -> S2
```

iff

```
T2 inh T1
S1 inh S2
```

In other words, we assume higher-order type inheritance is countervariant. The reason is that, if R1 = T1 -> S1 is to be a special case of R2 = T2 ->

S2, then one has to be able to use the latter everywhere one uses the former. This means that any input R2 takes, has to also be taken by R1 (hence T2 inherits from T1). And it means that the outputs R2 gives must be able to be accepted by any function that accepts outputs of R1 (hence S1 inherits from S2).

This type of issue comes up in programming language design fairly frequently, and there are a number of research papers debating the pros and cons of countervariance versus covariance for complex type inheritance. However, for the purpose of schema type inheritance in CogPrime , the greater logical consistency of the countervariance approach holds sway.

For instance, in this approach, `INT -> INT` is not a subtype of `NO -> INT` (where `NO` denotes `FLOAT`), because `NO -> INT` is the type that includes all functions which take a real and return an int, and an `INT -> INT` does not take a real. Rather, the containment is the other way around: every `NO -> INT` function is an example of an `INT -> INT` function. For example, consider the `NO -> INT` that takes every real number and rounds it up to the nearest integer. Considered as an `INT -> INT` function, this is simply the identity function: it is the function that takes an integer and rounds it up to the nearest integer.

Of course, tupling of types is different, it's covariant. If one has an ordered pair whose elements are of different types, say `(T1, T2)`, then we have

```
(T1 , S1) inh (T2, S2)
```

iff

```
T1 inh T2
S1 inh S2
```

As a mnemonic formula, we may say

```
(general -> specific ) inherits from (specific -> general)

(specific, specific) inherits from (general, general)
```

In schema learning, we will also have use for abstract type constructions, such as

```
(T1, T2) where T1 inherits from T2
```

Notationally, we will refer to variable types as Xv1, Xv2, etc., and then denote the inheritance relationships by using numerical indices, e.g. using

```
[1 inh 2]
```

to denote that

```
Xv1 inh Xv2
```

So for example,

```
( INT, VOID ) inh ( Xv1 , Xv2 )
```

is true, because there are no restrictions on the variable types, and we can just assign Xv1 = INT, Xv2 = VOID.

On the other hand,

```
( INT, VOID ) inh ( Xv1, Xv2 ), [ 1 inh 2 ]
```

is false because the restriction Xv1 inh Xv2 is imposed, but it's not true that INT inh VOID.

The following list gives some examples of type inheritance, using the elementary types INT, FLOAT (FL), NUMBER (NO), CHAR and STRING (STR), with the elementary type inheritance relationships

- INT inh NUMBER
- FLOAT inh NUMBER
- CHAR inh STRING
- (NO -> FL) inh (INT -> FL)
- (FL -> INT) inh (FL -> NO)
- ((INT -> FL) -> (FL -> INT)) inh ((NO -> FL) -> (FL -> NO)) WIKISOURCE:AbstractSchemaManipulation

20.3.5 Advanced Schema Manipulation

Now we describe some special schema for manipulating schema, which seem to be very useful in certain contexts.

20.3.5.1 Listification

First, there are two ways to represent n-ary relations in CogPrime 's Atom level knowledge representation language: using lists as in

```
f_list (x1, ..., xn)
```

or using currying as in

```
f_curry x1 ... xn
```

To make conversion between list and curried forms easier, we have chosen to introduce special schema (combinators) just for this purpose:

```
listify f = f_list so that f_list (x1, ..., xn ) = f x1 ... xn

unlistify listify f = f
```

For instance

```
kick_curry Ben Ken
```

denotes

```
(kick_curry Ben) Ken
```

which means that kick is applied to the argument Ben to yield a predicate schema applied to Ken. This is the curried style. The list style is

```
kick_List (Ben, Ken)
```

where kick is viewed as taking as an argument the List (Ben, Ken). The conversion between the two is done by

```
listify kick_curry = kick_list

unlistify kick_list = kick_curry
```

As a more detailed example of unlistification, let us utilize a simple mathematical example, the function $(X - 1)^2$. If we use the notations - and *pow* to denote SchemaNodes embodying the corresponding operations, then this formula may be written in variable-free node-and-link form as

```
ExOutLink
    pow
    ListLink
        ExOutLink
            -
            ListLink
                X
                1
        2
```

But to get rid of the nasty variable X, we need to first unlistify the functions *pow* and -, and then apply the C and B combinators a couple times to move the variable X to the front. The B combinator (see Combinatory Logic REF) is recalled below:

```
B f g h = f (g h)
```

This is accomplished as follows (using the standard convention of left-associativity for the application operator, denoted @ in the tree representation given in Section 20.3.0.6)

```
pow(-(x, 1), 2)
unlistify pow (-(x, 1) 2)
C (unlistify pow) 2 (-(x,1))
C (unlistify pow) 2 ((unlistify -) x 1)
C (unlistify pow) 2 (C (unlistify -) 1 x)
B (C (unlistify pow) 2) (C (unlistify -) 1) x
```

yielding the final schema

```
B (C (unlistify pow) 2) (C (unlistify -) 1)
```

By the way, a variable-free representation of this schema in CogPrime would look like

```
ExOutLink
    ExOutLink
```

```
        B
        ExOutLink
            ExOutLink
                C
                ExOutLink
                    unlistify
                    pow
            2
    ExOutLink
        ExOutLink
            C
            ExOutLink
                unlistify
                -
        1
```

The main thing to be observed is that the introduction of these extra schema lets us remove the variable X. The size of the schema is increased slightly in this case, but only slightly - an increase that is well-justified by the elimination of the many difficulties that explicit variables would bring to the system. Furthermore, there is a shorter rendition which looks like

```
ExOutLink
    ExOutLink
        B
        ExOutLink
            ExOutLink
                C
                pow_curried
            2
    ExOutLink
        ExOutLink
            C
            -_curried
        1
```

This rendition uses alternate variants of - and pow schema, labeled `-_curried` and `pow_curried`, which do not act on lists but are *curried* in the manner of combinatory logic and Haskell. It is 13 lines whereas the variable-bearing version is 9 lines, a minor increase in length that brings a lot of operational simplification.

20.3.5.2 Argument Permutation

In dealing with List relationships, there will sometimes be use for an argument-permutation operator, let us call it P, defined as follows

```
(P p f) (v1, ..., vn ) = f (p (v1, ..., vn ))
```

where p is a permutation on n letters. This deals with the case where we want to say, for instance that

```
Equivalence parent(x,y) child(y,x)
```

Instead of positing variable names x and y that span the two relations parent(x,y) and child(y,x), what we can instead say in this example is

```
Equivalence parent (P {2,1} child)
```

For the case of two-argument functions, argument permutation is basically doing on the list level what the C combinator does in the curried function domain. On the other hand, in the case of n-argument functions with n>2, argument permutation doesn't correspond to any of the standard combinators.

Finally, let's conclude with a similar example in a more standard predicate logic notation, involving both combinators and the permutation argument operator introduced above. We will translate the variable-laden predicate

```
likes(y,x) AND likes(x,y)
```

into the equivalent combinatory logic tree. Let us first recall the combinator S whose function is to distribute an argument over two terms.

```
S f g x = (f x) (g x)
```

Assume that the two inputs are going to be given to us as a list. Now, the combinatory logic representation of this is

```
S (B AND (B (P {2,1} likes))) likes
```

We now show how this would be evaluated to produce the correct expression:

```
S (B AND (B (P {2,1} likes))) likes (x,y)
```

S gets evaluated first, to produce

```
(B AND (B (P {2,1} likes)) (x,y)) (likes (x,y))
```

now the first B

```
AND ((B (P {2,1} likes)) (x,y)) (likes (x,y))
```

now the second one

```
AND ((P {2,1} likes) (x,y)) (likes (x,y))
```

now P

```
AND (likes (y,x)) (likes (x,y))
```

which is what we wanted.

Chapter 21
Representing Procedural Knowledge

21.1 Introduction

We now turn to CogPrime 's representation and manipulation of *procedural knowledge*. In a sense this is the most fundamental kind of knowledge – since intelligence is most directly about action selection, and it is procedures which generate actions.

CogPrime involves multiple representations for procedures, including procedure maps and (for sensorimotor procedures) neural nets or similar structures. Its most basic procedural knowledge representation, however, is the *program*. The choice to use programs to represent procedures was made after considerable reflection – they are not of course the only choice, as other representations such as recurrent neural networks possess identical representational power, and are preferable in some regards (e.g. resilience with respect to damage). Ultimately, however, we chose programs due to their consilience with the software and hardware underlying CogPrime (and every other current AI program). CogPrime is a program, current computers and operating systems are optimized for executing and manipulating programs; and we humans now have many tools for formally and informally analyzing and reasoning about programs. The human brain probably doesn't represent most procedures as programs in any simple sense, but CogPrime is not intended to be an emulation of the human brain. So, the representation of programs as procedures is one major case where CogPrime deviates from the human cognitive architecture in the interest of more effectively exploiting its own hardware and software infrastructure.

CogPrime represents procedures as programs in an internal programming language called "Combo." While Combo has a textual representation, described online at the OpenCog wiki, this isn't one of its more important aspects (and may be redesigned slightly or wholly without affecting system intelligence or architecture); the essence of Combo programs lies in their tree representation not their text representation. One could fairly consider Combo

as a dialect of LISP, although it's not equivalent to any standard dialect, and it hasn't particularly been developed with this in mind. In this chapter we discuss the key concepts underlying the Combo approach to program representation, seeking to make clear at each step the motivations for doing things in the manner proposed.

In terms of the overall CogPrime architecture diagram given in Chapter 6 of Part 1, this chapter is about the box labeled "Procedure Repository." The latter, in OpenCog, is a specialized component connected to the AtomSpace, storing Combo tree representations of programs; each program in the repository is linked to a SchemaNode in the AtomSpace, ensuring full connectivity between procedural and declarative knowledge.

21.2 Representing Programs

What is a "program" anyway? What distinguishes a program from an arbitrary representation of a procedure?

The essence of programmatic representations is that they are well-specified, compact, combinatorial, and hierarchical:

- *Well-specified:* unlike sentences in natural language, programs are unambiguous; two distinct programs can be precisely equivalent.
- *Compact:* programs allow us to compress data on the basis of their regularities. Accordingly, for the purposes of this chapter, we do not consider overly constrained representations such as the well-known conjunctive and disjunctive normal forms for Boolean formulae to be programmatic. Although they can express any Boolean function (data), they dramatically limit the range of data that can be expressed compactly, compared to unrestricted Boolean formulae.
- *Combinatorial:* programs access the results of running other programs (e.g. via function application), as well as delete, duplicate, and rearrange these results (e.g., via variables or combinators).
- *Hierarchical:* programs have intrinsic hierarchical organization, and may be decomposed into subprograms.

Eric Baum has advanced a theory "under which one understands a problem when one has mental programs that can solve it and many naturally occurring variations" [Bau06]. In this perspective – which we find an agreeable way to think about procedural knowledge, though perhaps an overly limited perspective on mind as a whole – one of the primary goals of artificial general intelligence is systems that can represent, learn, and reason about such programs [Bau06, Bau04]. Furthermore, integrative AGI systems such as CogPrime may contain subsystems operating on programmatic representations. Would-be AGI systems with no direct support for programmatic representation will clearly need to represent procedures and procedural abstractions

somehow. Alternatives such as recurrent neural networks have serious downsides, including opacity and inefficiency, but also have their advantages (e.g. recurrent neural nets can be robust with regard to damage, and learnable via biologically plausible algorithms).

Note that the problem of how to represent programs for an AGI system dissolves in the unrealistic case of unbounded computational resources. The solution is algorithmic information theory [Cha08], extended recently to the case of sequential decision theory [Hut05]. The latter work defines the universal algorithmic agent AIXI, which in effect simulates all possible programs that are in agreement with the agent's set of observations. While AIXI is uncomputable, the related agent AIXI^{tl} may be computed, and is superior to any other agent bounded by time t and space l [**?**]. The choice of a representational language for programs[1] is of no consequence, as it will merely introduce a bias that will disappear within a constant number of time steps.[2]

Our goal in this chapter is to provide practical techniques for approximating the ideal provided by algorithmic probability, based on what Pei Wang has termed the *assumption of insufficient knowledge and resources* [Wan06], and assuming an AGI architecture that's at least vaguely humanlike in nature, and operates largely in everyday human environments, but uses programs to represent many procedures. Given these assumptions, how programs are represented is of paramount importance, as we shall see in the next two sections, where we give a conceptual formulation of what we mean by *tractable program representations*, and introduce tools for formalizing such representations. The fourth section delves into effective techniques for representing programs. A key concept throughout is *syntactic-semantic correlation*, meaning that programs which are similar on the syntactic level, within certain constraints will tend to also be similar in terms of their behavior (i.e. on the semantic level). Lastly, the fifth section changes direction a bit and discusses the translation of programmatic structure into declarative form for the purposes of logical inference.

In the future, we will experimentally validate that these normal forms and heuristic transformations *do* in fact increase the syntactic-semantic correlation in program spaces, as has been shown so far only in the Boolean case. We would also like to explore the extent to which even stronger correlation, and additional tractability properties, can be observed when realistic probabilistic constraints on "natural" environment and task spaces are imposed.

The importance of a good programmatic representation of procedural knowledge becomes quite clear when one thinks about it in terms of the Mind-World Correspondence Principle introduced in Chapter 10. That principle states, roughly, that transition paths between world-states should map naturally onto transition paths between mind-states. This suggests that there should be a natural, smooth mapping between *real-world action series* and

[1] As well as a language for proofs in the case of AIXI^{tl}.

[2] The universal distribution converges quickly [**?**].

the corresponding *series of internal states*. Where internal states are driven by explicitly given programs, this means that the transitions between internal program states should nicely mirror transitions between the states of the real world as it interacts with the system controlled by the program. The extent to which this is true will depend on the specifics of the programming language – and it will be true for a much greater extent, on the whole, if the programming language displays high syntactic-semantic correlation for behaviors that commonly occur when the program is used to control the system in the real world. So, the various technical issues mentioned above and considered below, regarding the qualities desired in a programmatic representation, are merely the manifestation of the general Mind-World Correspondence Principle in the context of procedural knowledge, under the assumption that procedures are represented as programs. The material in this chapter may be viewed as an approach to ensuring the validity of the Mind-World Correspondence principle for programmatically-represented procedural knowledge, for CogPrime systems concerned with achieving humanly meaningful goals in everyday human environments.

21.3 Representational Challenges

Despite the advantages outlined in the previous section, there are a number of challenges in working with programmatic representations:

- **Open-endedness** – in contrast to some other knowledge representations current in machine learning, programs vary in size and "shape", and there is no obvious problem-independent upper bound on program size. This makes it difficult to represent programs as points in a fixed-dimensional space, or to learn programs with algorithms that assume such a space.
- **Over-representation** – often, syntactically distinct programs will be semantically identical (i.e. represent the same underlying behavior or functional mapping). Lacking prior knowledge, many algorithms will inefficiently sample semantically identical programs repeatedly [Loo07a, GBK04].
- **Chaotic Execution** – programs that are very similar, syntactically, may be very different, semantically. This presents difficulties for many heuristic search algorithms, which require syntactic and semantic distance to be correlated [Loo07b, TVCC05].
- **High resource-variance** – programs in the same space vary greatly in the space and time they require to execute.

It's easy to see how the latter two issues may present a challenge for mind-world correspondence! Chaotic execution makes it hard to predict whether a program will indeed manifest state-sequences mapping nicely to a corresponding world-sequences; and high resource-variance makes it hard to pre-

dict whether, for a given program, this sort of mapping can be achieved for relevant goals given available resources.

Based on these concerns, it is no surprise that search over program spaces quickly succumbs to combinatorial explosion, and that heuristic search methods are sometimes no better than random sampling [LP02]. However, alternative representations of procedures also have their difficulties, and so far we feel the thornier aspects of programmatic representation are generally an acceptable price to pay in light of the advantages.

For some special cases in CogPrime we have made a different choice – e.g. when we use DeSTIN for sensory perception (see Chapter 28 we utilize a more specialized representation comprising a hierarchical network of more specialized elements. DeSTIN doesn't have problems with resource variance or chaotic execution, though it does suffer from over-representation. It is not very open-ended, which helps increase its efficiency in the perceptual processing domain, but may limit its applicability to more abstract cognition. In short we feel that, for general representation of cognitive procedures, the benefits of programmatic representation outweigh the costs; but for some special cases such as low-level perception and motor procedures, this may not be true and one may do better to opt for a more specialized, more rigid but less problematic representation.

It would be possible to modify CogPrime to use, say, recurrent neural nets for procedure representation, rather than programs in an explicit language. However, this would rate as a rather major change in the architecture, and would cause multiple problems in other aspects of the system. For example, programs are reasonably straightforward to reason about using PLN inference, whereas reasoning about the internals of recurrent neural nets is drastically more problematic, though not impossible. The choice of a procedure representation approach for CogPrime has been made considering not only procedural knowledge in itself, but the interaction of procedural knowledge with other sorts of knowledge. This reflects the general synergetic nature of the CogPrime design.

There are also various computation-theoretic issues regarding programs; however, we suspect these are not particularly relevant to the task of creating human-level AGI, though they may rear their heads when one gets into the domain of super-human, profoundly self-modifying AGI systems. For instance, in the context of the the difficulties caused by over-representation and high resource-variance, one might observe that determinations of e.g. programmatic equivalence for the former, and e.g. halting behavior for the latter, are uncomputable. But we feel that, given the assumption of insufficient knowledge and resources, these concerns dissolve into the larger issue of computational intractability and the need for efficient heuristics. Determining the equivalence of two Boolean formulae over 500 variables by computing and comparing their truth tables is trivial from a computability standpoint, but, in the words of Leonid Levin, "only math nerds would call 2^{500} finite" [Lev94].

Similarly, a program that never terminates is a special case of a program that runs too slowly to be of interest to us.

One of the key ideas underlying our treatment of programmatic knowledge is that, in order to tractably learn and reason about programs, an AI system must have prior knowledge of programming language semantics. That is, in the approach we advocate, the mechanism whereby programs are executed is assumed known *a priori*, and assumed to remain constant across many problems. One may then craft AI methods that make specific use of the programming language semantics, in various ways. Of course in the long run a sufficiently powerful AGI system could modify these aspects of its procedural knowledge representation; but in that case, according to our approach, it would also need to modify various aspects of its procedure learning and reasoning code accordingly.

Specifically, we propose to exploit prior knowledge about program structure via enforcing programs to be represented in normal forms that preserve their hierarchical structure, and to be heuristically simplified based on reduction rules. Accordingly, one formally equivalent programming language may be preferred over another by virtue of making these reductions and transformations more explicit and concise to describe and to implement. The current OpenCogPrime system uses a simple LISP-like language called Combo (which takes both tree form and textual form) to represent procedures, but this is not critical; the main point is using some language or language variant that is "tractable" in the sense of providing a context in which the semantically useful reductions and transformations we've identified are naturally expressible and easily usable.

21.4 What Makes a Representation Tractable?

Creating a comprehensive formalization of the notion of a *tractable program representation* would constitute a significant achievement; and we will not answer that summons here. We will, however, take a step in that direction by enunciating a set of positive principles for tractable program representations, corresponding closely to the list of representational challenges above. While the discussion in this section is essentially conceptual rather than formal, we will use a bit of notation to ensure clarity of expression; S to denote a space of programmatic functions of the same type (e.g. all pure *Lisp* λ-expressions mapping from lists to numbers), and B to denote a metric space of *behaviors*.

In the case of a deterministic, side-effect-free program, execution maps from programs in S to points in B, which will have separate dimensions for the function's output across various inputs of interest, as well as dimensions corresponding to the time and space costs of executing the program. In the case of a program that interacts with an external environment, or is intrinsically nondeterministic, execution will map from S to probability distributions

over points in B, which will contain additional dimensions for any side-effects of interest that programs in S might have. Note the distinction between *syntactic distance*, measured as e.g. tree-edit distance between programs in S, and *semantic distance*, measured between program's corresponding points in or probability distributions over B. We assume that semantic distance accurately quantifies our preferences in terms of a weighting on the dimensions of B; i.e., if variation along some axis is of great interest, our metric for semantic distance should reflect this.

Let $\mathcal{P}$ be a probability distribution over B that describes our knowledge of what sorts of problems we expect to encounter, let $R(n) \subseteq S$ be all the programs in our representation with (syntactic) size no greater than n. We will say that $R(n)$ *d-covers* the pair $(B, \mathcal{P})$ to extent p if the probability that, for a random behavior $b \in B$ chosen according to $\mathcal{P}$, there is some program in R whose behavior is within semantic distance d of b, is greater than or equal to p. Then, some among the various properties of tractability that seem important based on the above discussion are as follows:

- for fixed d, p quickly goes to 1 as n increases,
- for fixed p, d quickly goes to 0 as n increases,
- for fixed d and p, the minimal n needed for $R(n)$ to *d-cover* $(B, \mathcal{P})$ to extent p should be as small as possible,
- ceteris paribus, syntactic and semantic distance (measured according to $\mathcal{P}$) are highly correlated.

This is closely related to the Mind-Brain Correspondence Principle articulated in Chapter 10, and to the geometric formulation of cognitive synergy posited in Appendix B. Syntactic distance has to do with distance along paths in mind-space related to formal program structures, and syntactic distance has to do with distance along paths in mind-space and world-space corresponding to the record of the program's actual behavior. If syntax-semantics correlation failed, then there would be paths through mind-space (related to formal program structures) that were poorly matched to their closest corresponding paths through the rest of mind-space and world-space, hence causing a failure (or significant diminution) of cognitive synergy and mind-world correspondence.

Since execution time and memory usage considerations may be incorporated into the definition of program behavior, minimizing chaotic execution and managing resource variance emerges conceptually here as subcases of maximizing correlation between syntactic and semantic distance. Minimizing over-representation follows from the desire for small program size: roughly speaking the less over-representation there is, the smaller average program size can be achieved.

In some cases one can achieve fairly strong results about tractability of representations without any special assumptions about $\mathcal{P}$: for example in prior work we have shown that adoption of an appropriate hierarchical normal form can generically increase correlation between syntactic and semantic distance

in the space of Boolean functions [?, Loo07b]. In this case we may say that we have a *generically tractable* representation. However, to achieve tractable representation of more complex programs, some fairly strong assumptions about $\mathcal{P}$ will be necessary. This should not be philosophically disturbing, since it's clear that human intelligence has evolved in a manner strongly conditioned by certain classes of environments; and similarly, what we need to do to create a viable program representation system for pragmatic AGI usage, is to achieve tractability relative to the distribution $\mathcal{P}$ corresponding to the actual problems the AGI is going to need to solve. Formalizing the distributions $\mathcal{P}$ of real-world interest is a difficult problem, and one we will not address here (recall the related, informal discussions of Chapter 9, where we considered the various important peculiarities of the human everyday world). However, we hypothesize that the representations presented in the following section may be tractable to a significant extent irrespective of $\mathcal{P}$,[3] and even more powerfully tractable with respect to this as-yet unformalized distribution. As weak evidence in favor of this hypothesis, we note that many of the representations presented have proved useful so far in various narrow problem-solving situations.

21.5 The Combo Language

The current version of OpenCogPrime uses a simple language called Combo, which is an example of a language in which the transformations we consider important for AGI-focused program representation are relatively simple and natural. Here we illustrate the Combo language by example, referring the reader to the OpenCog wiki site for a formal presentation.

The main use of the Combo language in OpenCog is behind-the-scenes, i.e. using tree representations of Combo programs; but there is also a human-readable syntax, and an interpreter that allows humans to write Combo programs when needed. The main use of Combo, however, is not for human-coded programs, but rather for programs that are learned via various AI methods.

In Combo all expressions are in prefix form like LISP, but the left parenthesis is placed after the operator instead of before, for example:

- `+(4 5)`
  is a 0-ari expression that returns $4 + 5$
- `and(#1 0<(#2))`
  is a binary expression of type $bool \times float \mapsto bool$ that returns true if and only if the first input is true and the second input positive. $\#n$ designates the n-th input.
- `fact(1) := if(0<(#1) *(#1 fact(+(#1 -1))) 1)`
  is a recursive definition of factorial.

[3] Specifically, with only weak biases that prefer smaller and faster programs with hierarchical decompositions.

- `and_seq(goto(stick) grab(stick) goto(owner) drop)`

is a 0-ari expression with side effects, it evaluates a sequence of actions until completion or failure of one of them. Each action is executed in the environment the agent is connected to and returns *action_success* upon success or *action_failure* otherwise. The action sequence returns *action_success* if it completes or *action_failure* if it does not.

- ```
  if(near(owner self)
      lick(owner)
      and_seq(goto(owner) wag)
  ```

is a 0-ary expression with side effects; it means that if at the time of its evaluation the agent referred as self (here a virtual pet) is near its owner then lick him/her, otherwise go to the owner and wag the tail.

21.6 Normal Forms Postulated to Provide Tractable Representations

We now present a series of normal forms for programs, postulated to provide tractable representations in the contexts relevant to human-level, roughly human-like general intelligence.

21.6.1 A Simple Type System

We use a simple type system to distinguish between the various normal forms introduced below. This is necessary to convey the minimal information needed to correctly apply the basic functions in our canonical forms. Various systems and applications may of course augment these with additional type information, up to and including the satisfaction of arbitrary predicates (e.g. a type for prime numbers). This can be overlaid on top of our minimalist system to convey additional bias in selecting which transformations to apply, and introducing constraints as necessary. For instance, a call to a function expecting a prime number, called with a potentially composite argument, may be wrapped in a conditional testing the argument's primality. A similar technique is used in the normal form for functions to deal with list arguments that may be empty.

Normal forms are provided for *Boolean* and *number* primitive types, and the following parametrized types:

- list types, $list_T$, where T is any type,
- tuple types, $tuple_{T_1,T_2,\ldots T_N}$, where all T_i are types, and N is a positive natural number,
- enum types, $\{s_1, s_2, \ldots s_N\}$, where N is a positive number and all s_i are unique identifiers,
  ```

- function types $T_1, T_2, \ldots T_N \rightarrow O$, where O and all T_i are types,
- action result types.

A list of type $list_T$ is an ordered sequence of any number of elements, all of which must have type T. A tuple of type $tuple_{T_1,T_2,\ldots T_N}$ is an ordered sequence of exactly N elements, where every ith element is of type T_i. An enum of type $\{s_1, s_2, \ldots s_N\}$ is some element s_i from the set. Action result types concern side-effectful interaction with some world external to the system (but perhaps simulated, of course), and will be described in detail in their subsection below. Other types may certainly be added at a later date, but we believe that those listed above provide sufficient expressive power to conveniently encompass a wide range of programs, and serve as a compelling proof of concept.

The normal form for a type T is a set of elementary functions with codomain T, a set of constants of type T, and a tree grammar. Internal nodes for expressions described by the grammar are elementary functions, and leaves are either U_{var} or $U_{constant}$, where U is some type (often $U = T$).

Sentences in a normal form grammar may be transformed into normal form expressions. The set of expressions that may be generated is a function of a set of bound variables and a set of external functions that must be provided (both bound variables and external functions are typed). The transformation is as follows:

- $T_{constant}$ leaves are replaced with constants of type T,
- T_{var} leaves are replaced with either bound variables matching type T, or expressions of the form $f(expr_1, expr_2, \ldots expr_M)$, where f is an external function of type $T_1, T_2, \ldots T_M \rightarrow T$, and each $expr_i$ is a normal form expression of type T_i (given the available bound variables and external functions).

21.6.2 Boolean Normal Form

The elementary functions are *and*, *or*, and *not*. The constants are $\{true, false\}$. The grammar is:

```
bool_root = or_form  | and_form | literal  | bool_constant
literal   = bool_var | not( bool_var )
or_form   = or( {and_form | literal}{2,} )
and_form  = and( {or_form | literal}{2,} ) .
```

The construct `foo{x,}` refers to `x` or more matches of `foo` (e.g. `{x | y}{2,}` is two or more items in sequences where each item is either an `x` or a `y`).

21.6.3 Number Normal Form

The elementary functions are $*$ (times) and $+$ (plus). The constants are some subset of the rationals (e.g. those with IEEE single-precision floating-point representations). The grammar is:

```
num_root   = times_form | plus_form | num_constant | num_var
times_form = *( {num_constant | plus_form} plus_form{1,} )
           | num_var
plus_form  = +( {num_constant | times_form} times_form{1,} )
           | num_var
```

21.6.4 List Normal Form

For list types $list_T$, the elementary functions are *list* (an n-ary list constructor) and *append*. The only constant is the empty list (*nil*). The grammar is:

```
list_T_root = append_form | list_form | list_T_var
            | list_T_constant
append_form = append( {list_form | list_T_var}{2,} )
list_form   = list( T_root{1,} )
```

21.6.5 Tuple Normal Form

For tuple types $tuple_{T_1,T_2,\ldots T_N}$, the only elementary function is the tuple constructor (*tuple*). The constants are

$$T_1\texttt{\_constant} \times T_2\texttt{\_constant} \times \cdots \times T_N\texttt{\_constant}$$

The normal form is either a constant, a var, or

$$\texttt{tuple(}\ T_1\texttt{\_root}\ \ T_2\texttt{\_root}\ \ldots\ T_N\texttt{\_root}\ \texttt{)}$$

21.6.6 Enum Normal Form

Enums are atomic tokens with no internal structure - accordingly, there are no elementary functions. The constants for the enum $\{s_1, s_2, \ldots s_N\}$ are the s_is. The normal form is either a constant or a variable.

21.6.7 Function Normal Form

For $T_1, T_2, \ldots T_N \rightarrow O$, the normal form is a lambda-expression of arity N whose body is of type O. The list of variable names for the lambda-expression is not a "proper" argument - it does not have a normal form of its own. Assuming that none of the T_is is a list type, the body of the lambda-expression is simply in the normal form for type O (with the possibility of the lambda-expressions arguments appearing with their appropriate types). If one or more T_is are list types, then the body is a call to the *split* function with all arguments are in normal form.

Split is a family of functions with type signatures

$$(T_1, list_{T_1}, T_2, list_{T_2}, \ldots T_k, list_{T_k} \rightarrow O),$$
$$tuple_{list_{T_1},O}, tuple_{list_{T_2},O}, \ldots tuple_{list_{T_k},O} \rightarrow O.$$

To evaluate $split(f, tuple(l_1, o_1), tuple(l_2, o_2), \ldots\ tuple(l_k, o_k))$, the list arguments $l_1, l_2, \ldots l_k$ are examined sequentially. If some l_i is found that is empty, then the result is the corresponding value o_i. If all l_i are nonempty, we deconstruct each of them into $x_i : xs_i$, where x_i is the first element of the list and xs_i is the rest. The result is then $f(x_1, xs_1, x_2, xs_2, \ldots x_k, xs_k)$. The *split* function thus acts as an implicit case statement to deconstruct lists only if they are nonempty.

21.6.8 Action Result Normal Form

An action result type *act* corresponds to the result of taking an action in some world. Every action result type has a corresponding world type, *world*. Associated with action results and worlds are two special sorts of functions.

- *Perceptions* - functions that take a *world* as their first argument and regular (non-world and non-action-result) types as their remaining arguments, and return regular types. Unlike other function types, the result of evaluating a perception call may be different at different times, because the world will have different configurations at different times.
- *Actions* - functions that take a *world* as their first argument and regular types as their remaining arguments, and return action results (of the type associated with the type of their world argument). As with perceptions, the result of evaluating an action call may be different at different times. Furthermore, actions may have *side effects* in the associated world that they are called in. Thus, unlike any other sort of function, actions *must* be evaluated, even if their return values are ignored.

Other sorts of functions acting on worlds (e.g. ones that take multiple worlds as arguments) are disallowed.

Note that an action result expression cannot appear nested inside an expression of any other type. Consequently, there is no way to convert e.g. an action result to a Boolean, although conversion in the opposite direction is permitted. This is required because mathematical operations in our language have classical mathematical semantics; x *and* y must equal y *and* x, which will not generally be the case if x or y can have side-effects. Instead, there are special sequential versions of logical functions which may be used instead.

The elementary functions for action result types are and_{seq} (sequential and, equivalent to C's short-circuiting `&&`), or_{seq} (sequential or, equivalent to C's short-circuiting `||`), and $fails$ (negates success to failure and vice versa). The constants may vary from type to type but must at least contain *success* and $failure$, indicating absolute success/failure in execution.[4] The normal form is as follows:

```
act_root    = orseq_form | andseq_form | seqlit
seqlit      = act | fails( act )
act         = act_constant | act_var
orseq_form  = orseq( {andseq_form | seqlit}{2,} )
andseq_form = andseq( {orseq_form | seqlit}{2,} )
```

21.7 Program Transformations

A program transformation is any type-preserving mapping from expressions to expressions. Transformations may be guaranteed to preserve semantics. When doing program evolution there is an intermediate category of fitness preserving transformations that may alter semantics, but not fitness. In general, the only way that fitness preserving transformations will be uncovered is by scoring programs that have had their semantics potentially transformed to determine their fitness, which what most fitness function does. On the other hand if the fitness function is encompassed in the program itself, so a candidate directly outputs the fitness itself, then only preserving semantics transformations are needed.

21.7.1 Reductions

These are semantics preserving transformations that do not increase some size measure (typically number of symbols), and are idempotent. For example, $and(x, x, y) \rightarrow and(x, y)$ is a reduction for Boolean expressions. A set of *canonical reductions* is defined for every type that has a normal form. For numerical functions, the simplifier in a computer algebra system may be used.

[4] A $do(arg_1, arg_2, \ldots arg_N)$ statement (known as *progn* in Lisp), which evaluates its arguments sequentially regardless of success or failure, is equivalent to $and_{seq}(or_{seq}(arg_1, success), or_{seq}(arg_2, success), \ldots\ or_{seq}(arg_N, success))$.

The full list of reductions is omitted in for brevity. An expression is *reduced* if it maps to itself under all canonical reductions for its type, and all of its children are reduced.

Another important set of reductions are the *compressive abstractions*, which reduce or keep constant the size of expressions by introducing new functions. Consider

```
list(*(+(a p q) r)
     *(+(b p q) r)
     *(+(c p q) r))
```

which contains 19 symbols. Transforming this to

```
f(x) = *(+(x p q) r)
list(f(a) f(b) f(c))
```

reduces the total number of symbols to 15. One can generalize this notion to consider compressive abstractions across a set of programs. Compressive abstractions appear to be rather expensive to uncover, although not prohibitively so, the computation may easily be parallelized and may rely heavily on subtree mining [TODO REF].

21.7.1.1 A Simple Example of Reduction

We now give a simple example of how CogPrime 's reduction engine can transform a program into a semantically equivalent but shorter one.

Consider the following program and the chain of reduction:

1. We start with the expression

   ```
   if(P and_seq(if(P A B) B) and_seq(A B))
   ```

2. A reduction rule permits to reduce the conditional `if(P A B)` to `if(true A B)`. Indeed if `P` is true, then the first branch is evaluated and `P` must still be true.

   ```
   if(P and_seq(if(true A B) B) and_seq(A B))
   ```

3. Then a rule can reduce `if(true A B)` to `A`.

   ```
   if(P and_seq(A B) and_seq(A B))
   ```

4. And finally another rule replaces the conditional by one of its branches since they are identical

   ```
   and_seq(A B)
   ```

Note that the reduced program is not only smaller (3 symbols instead of 11) but a bit faster too. Of course it is not generally true that smaller programs are faster but in the restricted context of our experiments it has often been the case.

21.7.2 Neutral Transformations

Semantics preserving transformations that are not reductions are not useful on their own - they can only have value when followed by transformations from some other class. They are thus more speculative than reductions, and more costly to consider. I will refer to these as *neutral transformations* [Ols95].

- **Abstraction** - given an expression E containing non-overlapping subexpressions $E_1, E_2, \ldots E_N$, let E' be E with all E_i replaced by the unbound variables v_i. Define the function $f(v_1, v_2, \ldots v_3) = E'$, and replace E with $f(E_1, E_2, \ldots E_N)$. Abstraction is distinct from compressive abstraction because only a single call to the new function f is introduced.[5]
- **Inverse abstraction** - replace a call to a user-defined function with the body of the function, with arguments instantiated (note that this can also be used to partially invert a compressive abstraction).
- **Distribution** - let E be a call to some function f, and let E' be an expression of E's ith argument that is a call to some function g, such that f is distributive over g's arguments, or a subset thereof. We shall refer to the actual arguments to g in these positions in E' as $x_1, x_2, \ldots x_n$. Now, let $D(F)$ be the function that is obtained by evaluating E with its ith argument (the one containing E') replaced with the expression F. Distribution is replacing E with E', and then replacing each x_j ($1 \leq j \leq n$) with $D(x_j)$. For example, consider

  ```
  +(x *(y if(cond a b)))
  ```

  Since both + and * are distributive over the result branches of *if*, there are two possible distribution transformations, giving the expressions

  ```
  if(cond +(x *(y a)) +(x *(y b)))
  +(x(if(cond *(y a) *(y b))))
  ```

- **Inverse distribution (factorization)** - the opposite of distribution. This is nearly a reduction; the exceptions are expressions such as $f(g(x))$, where f and g are mutually distributive.
- **Arity broadening** - given a function f, modify it to take an additional argument of some type. All calls to f must be correspondingly broadened to pass it an additional argument of the appropriate type.
- **List broadening**[6] - given a function f with some ith argument x of type T, modify f to instead take an argument y of type $list_T$, which gets split into $x : xs$. All calls to f with ith argument x' must be replaced by corresponding calls with ith argument $list(x')$.
- **Conditional insertion** - an expression x is replaced by $if(true, x, y)$, where y is some expression of the same type of x.

[5] In compressive abstraction there must be at least two calls in order to avoid increasing the number of symbols.

[6] Analogous tuple-broadening transformations may be defined as well, but are omitted for brevity.

As a technical note, action result expressions (which may cause side-effects) complicate neutral transformations. Specifically, abstractions and compressive abstractions must take their arguments lazily (i.e. not evaluate them before the function call itself is evaluated), in order to be neutral. Furthermore, distribution and inverse distribution may only be applied when f has no side-effects that will vary (e.g. be duplicated or halved) in the new expression, or affect the nested computation (e.g. change the result of a condition within a conditional). Another way to think about this issue is to consider the action result type as a lazy domain-specific language embedded within a pure functional language (where evaluation order is unspecified). Spector has performed an empirical study of the tradeoffs in lazy vs. eager function abstraction for program evolution [Spe96].

The number of neutral transformation applicable to any given program grows quickly with program size.[7] Furthermore, synthesis of complex programs and abstractions does not seem to be possible without them. Thus, a key hypothesis of any approach to AGI requiring significant program synthesis, without assuming the currently infeasible computational capacities required to brute-force the problem, is that the inductive bias to select promising neutral transformations can be learned and/or programmed. Referring back to the initial discussion of what constitutes a tractable representation, we speculate that perhaps, whereas well-chosen reductions are valuable for generically increasing program representation tractability, well-chosen neutral transformations will be valuable for increasing program representation tractability relative to distributions $\mathcal{P}$ to which the transformations have some (possibly subtle) relationship.

21.7.3 Non-Neutral Transformations

Non-neutral transformations are the general class defined by removal, replacement, and insertion of subexpressions, acting on expressions in normal form, and preserving the normal form property. Clearly these transformations are sufficient to convert any normal form expression into any other. What is desired is a subclass of the non-neutral transformations that is combinatorially complete, where each individual transformation is nonetheless a semantically small step.

The full set of transformations for Boolean expressions is given in [**?**]. For numerical expressions, the transcendental functions sin, log, and e^x are used to construct transformations. These obviate the need for division ($a/b = e^{log(a)-log(b)}$), and subtraction ($a - b = a + -1 * b$). For lists, transformations are based on insertion of new leaves (e.g. to append function calls), and "deepening" of the normal form by insertion of subclauses (see

[7] Exact calculations are given by Olsson [Ols95].

[?] for details). For tuples, we take the union of the transformations of all the subtypes. For other mixed-type expressions the union of the non-neutral transformations for all types must be considered as well. For enum types the only transformation is replacing one symbol with another. For function types, the transformations are based on function composition. For action result types, actions are inserted/removed/altered, akin to the treatment of Boolean literals for the Boolean type.

We propose an additional class of non-neutral transformations based on the marvelous *fold* function:

$$fold(f, v, l) = if(empty(l), v, f(first(l), fold(f, v, rest(l))))$$

With *fold* we can express a wide variety of iterative constructs, with guaranteed termination and a bias towards low computational complexity. In fact, fold allows us to represent exactly the primitive recursive functions [Hut99].

Even considering only this reduced space of possible transformations, in many cases there are still too many possible programs "nearby" some target to effectively consider all of them. For example many probabilistic model-building algorithms, such as learning the structure of a Bayesian network from data, can require time cubic in the number of variables (in this context each independent non-neutral transformation can correspond to a variable). Especially as the size of the programs we wish to learn grows, and as the number of typologically matching functions increases, there will be simply too many variables to consider each one intensively, let alone apply a quadratic-time algorithm.

To alleviate this scaling difficulty, we propose three techniques.

The first is to consider each potential variable (i.e. independent non-neutral transformation) to heuristically determine its usefulness in expressing constructive semantic variation. For example, a Boolean transformation that collapses the overall expression into a tautology is assumed to be useless.[8]

The second is heuristic coupling rules that allow us to calculate, for a pair of transformations, the expected utility of applying them in conjunction.

Finally, while *fold* is powerful, it may need to be augmented by other methods in order to provide tractable representation of complex programs that would normally be written using numerous variables with diverse scopes. One approach that we have explored involves application of [SMI97]'s ideas about *director strings as combinators*. In Sinot's approach, special program tree nodes are labeled with director strings, and special algebraic operators interrelate these strings. One then achieves the representational efficiency of local variables with diverse scopes, without needing to do any actual variable management. Reductions and other (non-)neutral transformation rules related to broadening and reducing variable scope may then be defined using the director string algebra.

[8] This is heuristic because such a transformation might be useful together with other transformations.

21.8 Interfacing Between Procedural and Declarative Knowledge

Finally, another critical aspect of procedural knowledge is its interfacing with declarative knowledge. We now discuss the referencing of declarative knowledge within procedures, and the referencing of the details of procedural knowledge within CogPrime 's declarative knowledge store.

21.8.1 Programs Manipulating Atoms

Above we have used Combo syntax implicitly, referring to Appendix **??** for the formal definitions. Now we introduce one additional, critical element of Combo syntax: the capability to explicitly reference declarative knowledge within procedures.

For this purpose Combo must contain the following types:

$$Atom, Node, Link, TruthValue, AtomType, AtomTable$$

Atom is the union of Node and Link.

So a type Node within a Combo program refers to a Node in CogPrime 's AtomTable. The mechanisms used to evaluate these entities during program evaluation are discussed in Chapter 25.

For example, suppose one wishes to write a Combo program that creates Atoms embodying the predicate-argument relationship $eats(cat, fish)$, represented

```
Evaluation eats (cat, fish)
```

aka

```
Evaluation
   eats
      List
         cat
         fish
```

To do this, one could say for instance,

```
new-link(EvaluationLink
         new-node(PredicateNode ``eats'')
         new-link(ListLink
                  new-node(ConceptNode ``cat'')
                  new-node(ConceptNode ``fish''))
         (new-stv .99 .99))
```

21.9 Declarative Representation of Procedures

Next, we consider the representation of program tree internals using declarative data structures. This is important if we want OCP to inferentially *understand* what goes on inside programs. In itself, it is more of a "bookkeeping" issue than a deep conceptual issue, however.

First, note that each of the entities that can live at an internal node of a program, can also live in its own Atom. For example, a number in a program tree corresponds to a NumberNode; an argument in a Combo program already corresponds to some Atom; and an operator in a program can be wrapped up in a SchemaNode all its own, and considered as a one-leaf program tree.

Thus, one can build a kind of virtual, distributed program tree by linking a number of ProcedureNodes (i.e. PredicateNodes or SchemaNodes) together. All one needs in order to achieve this is an analogue of the @ symbol (as defined in Section 20.3 of Chapter 20) for relating ProcedureNodes. This is provided by the ExecutionLink type, where

```
(ExecutionLink f g)
```

essentially means the same as

```
f g
```

in curried notation or

```
   @
  / \
 f   g
```

The same generalized evaluation rules used inside program trees may be thought of in terms of ExecutionLinks; formally, they are crisp ExtensionalImplicationLinks among ExecutionLinks.

Note that we are here using ExecutionLink as a curried function; that is, we are looking at (ExecutionLink f g) as a function that takes an argument x, where the truth value of

```
(ExecutionLink f g) x
```

represents the probability that executing f, on input g, will give output x.

One may then construct combinator expressions linking multiple ExecutionLinks together; these are the analogues of program trees.

For example, using ExecutionLinks, one equivalent of y = x + x^2 is:

```
Hypothetical
    SequentialAND
        ExecutionLink
            pow
            List v1 2
            v2
        ExecutionLink
            +
            List v1 v2
            v3
```

Here the $v1, v2, v3$ are variables which may be internally represented via combinators. This AND is sequential in case the evaluation order inside the program interpreter makes a difference.

As a practical matter, it seems there is no purpose to explicitly storing program trees in conjunction-of-ExecutionLinks form. The information in the ExecutionLink conjunct is already there in the program tree. However, the PLN reasoning system, when reasoning on program trees, may carry out this kind of expansion internally as part of its analytical process.

Section VII
The Cognitive Cycle

Chapter 22
Emotion, Motivation, Attention and Control

Co-authored with Zhenhua Cai

22.1 Introduction

This chapter begins the heart of the book: the part that explains *how the CogPrime design aims to implement roughly human-like general intelligence, at the human level and ultimately beyond.* First, here in Section VII we explain how CogPrime can be used to implement a simplistic animal-like agent without much learning: an agent that perceives, acts and remembers, and chooses actions that it thinks will achieve its goals; but doesn't do any sophisticated learning or reasoning or pattern recognition to help it better perceive, act, remember or figure out how to achieve its goals. We're not claiming CogPrime is the best way to implement such an animal-like agent, though we suggest it's not a bad way and depending on the complexity and nature of the desired behaviors, it *could* be the best way. We have simply chosen to split off the parts of CogPrime needed for animal-like behavior and present them first, prior to presenting the various "knowledge creation" (learning, reasoning and pattern recognition) methods that constitute the more innovative and interesting part of the design.

In Stan Franklin's terms, what we explain here in Section VII is how a basic *cognitive cycle* may be achieved within CogPrime . In that sense, the portion of CogPrime explained in this Sectionis somewhat similar to the parts of Stan's LIDA architecture that have currently been worked out in detail, and that . However, while LIDA has not yet been extended in detail (in theory or implementation) to handle advanced learning, cognition and language, those aspects of CogPrime *have* been developed and in fact constitute the largest portion of this book.

Looking back to the integrative diagram from Chapter 5, the cognitive cycle is mainly about integrating vaguely LIDA-like structures and mechanisms with heavily Psi-like structures and mechanisms – but doing so in a way that naturally links in with perception and action mechanisms "below," and more abstract and advanced learning mechanisms "above."

In terms of the general theory of general intelligence, the basic CogPrime cognitive cycle can be seen to have a foundational importance in biasing the CogPrime system toward the problem of controlling an agent in an environment requiring a variety of real-time and near-real-time responses based on a variety of kinds of knowledge. Due to its basis in human and animal cognition, the CogPrime cognitive cycle likely incorporates many useful biases in ways that are not immediately obvious, but that would become apparent if comparing intelligent agents controlled by such a cycle versus intelligent agents controlled via other means.

The cognitive cycle also provides a framework in which other cognitive processes, relating to various aspects of the goals and environments relevant to human-level general intelligence, may conveniently dynamically interoperate. The "Mind OS" aspect of the CogPrime architecture provides general mechanisms in which various cognitive processes may interoperate on a common knowledge store; the cognitive cycle goes further and provides a specific dynamical pattern in which multiple cognitive processes may intersect. Its effective operation places strong demands on the cognitive synergy between the various cognitive processes involved, but also provides a framework that encourages this cognitive synergy to develop and persist.

Finally, it should be stressed that the cognitive cycle is not all-powerful nor wholly pervasive in CogPrime 's dynamics. It's critical for the real-time interaction of a CogPrime -controlled agent with a virtual or physical world; but there may be many processes within CogPrime that most naturally operate outside such a cycle. For instance, humans will habitually do deep intellectual thinking (even something so abstract as mathematical theorem proving) within a cognitive cycle somewhat similar to the one they use for practical interaction with the external world. But, there's no reason that CogPrime systems need to be constrained in this way. Deviating from a cognitive cycle based dynamic may cause a CogPrime system to deviate further from human-likeness in its intelligence, but may also help it to perform better than humans on some tasks, e.g. tasks like scientific data analysis or mathematical theorem proving that benefit from styles of information processing that humans aren't particularly good at.

22.2 A Quick Look at Action Selection

We will begin our exposition of CogPrime 's cognitive cycle with a quick look at *action selection*. As Stan Franklin likes to point out, the essence of an intelligent *agent* is that it does things; it takes *actions*. The particular mechanisms of action selection in CogPrime are a bit involved and will be given in Chapter 24; in this chapter we will give the basic idea of the action selection mechanism and then explain how a variant of the Psi model (described in

Chapter 4 of Part 1 above) is used to handle *motivation* (emotions, drives, goals, etc.) in CogPrime , including the guidance of action selection.

The crux of CogPrime 's action selection mechanism is as follows

- the action selector chooses procedures that seem likely to help achieve important goals in the current context
  - *Example:* If the goal is to create a block structure that will surprise Bob, and there is plenty of time, one procedure worth choosing might be a memory search procedure for remembering situations involving Bob and physical structures. Alternately, if there isn't much time, one procedure worth choosing might be a procedure for building the base of a large structure – as this will give something to use as part of whatever structure is eventually created. Another procedure worth choosing might be one that greedily assembles structures from blocks without any particular design in mind.
- to support the action selector, the system builds implications of the form $Context\&Procedure \rightarrow Goal$, where Context is a predicate evaluated based on the agent's situation
  - *Example:* If Bob has asked the agent to do something, and it knows that Bob is very insistent on being obeyed, then implications such as
    - "Bob instructed to do X" and "do X" $\rightarrow$ "please Bob" $< .9, .9 >$

    will be utilized
  - *Example:* If the agent wants to make a tower taller, then implications such as
    - "T is a blocks structure" and "place block atop T" $\rightarrow$ "make T taller" $< .9, .9 >$

    will be utilized
- the truth values of these implications are evaluated based on experience and inference
  - *Example:* The above implication involving Bob could be evaluated based on experience, by assessing it against remembered episodes involving Bob giving instructions
  - *Example:* The same implication could be evaluated based on inference, using analogy to experiences with instructions from other individuals similar to Bob; or using things Bob has explicitly said, combined with knowledge that Bob's self-descriptions tend to be reasonably accurate
- Importance values are propagated between goals using economic attention allocation (and, inference is used to learn subgoals from existing goals)
  - *Example:* If Bob has told the agent to do X, and the agent has then derived (from the goal of pleasing Bob) the goal of doing X, then

the "please Bob" goal will direct some of its currency to the "do X" goal (which the latter goal can then pass to its subgoals, or spend on executing procedures)

These various processes are carried out in a manner orchestrated by Dorner's Psi model as refined by Joscha Bach (as reviewed in Chapter 4 above), which supplies (among other features)

- a specific theory regarding what "demands" should be used to spawn the top-level goals
- a set of (four) interrelated system parameters governing overall system state in a useful manner reminiscent of human and animal psychology
- a systematic theory of how various emotions (wholly or partially) emerge from more fundamental underlying phenomena

22.3 Psi in CogPrime

The basic concepts of the Psi approach to motivation, as reviewed in Chapter 4 of Part 1 above, are incorporated in CogPrime as follows (note that the following list includes many concepts that will be elaborated in more detail in later chapters):

- Demands are GroundedPredicateNodes (GPNs), i.e. Nodes that have their truth value computed at each time by some internal C++ code or some Combo procedure in the ProcedureRepository
  - *Examples:* Alertness, perceived novelty, internal novelty, reward from teachers, social stimulus
  - Humans and other animals have familiar demands such as hunger, thirst and excretion; to create an AGI closely emulating a human or (say) a dog one may wish to simulate these in one's AGI system as well
- Urges are also GPNs, with their truth values defined in terms of the truth values of the Nodes for corresponding Demands. However in CogPrime we have chosen the term "Ubergoal" instead of Urge, as this is more evocative of the role that these entities play in the system's dynamics (they are the top-level goals).
- Each system comes with a fixed set of Ubergoals (and only very advanced CogPrime systems will be able to modify their Ubergoals)
  - *Example:* Stay alert and alive now and in the future; experience and learn new things now and in the future; get reward from the teachers now and in the future; enjoy rich social interactions with other minds now and in the future

  - A more advanced CogPrime system could have abstract (but experientially grounded) ethical principles among its Ubergoals, e.g. an Ubergoal to promote joy, an Ubergoal to promote growth and an Ubergoal to promote choice, in accordance with the ethics described in [Goe06a]

- The ShortTermImportance of an Ubergoal indicates the urgency of the goal, so if the Demand corresponding to an Ubergoal is within its target range, then the Ubergoal will have zero STI. But all Ubergoals can be given maximal LTI to guarantee they don't get deleted.
  - *Example:* If the system is in an environment continually providing an adequate level of novelty (according to its Ubergoal), then the Ubergoal corresponding to external novelty with have low STI but high LTI. The system won't expend resources seeking novelty. But then, if the environment becomes more monotonous, the urgency of the external novelty goal will increase, and its STI will increase correspondingly, and resources will begin getting allocated toward improving the novelty of the stimuli received by the agent.
- Pleasure is a GPN, and its internal truth value computing program compares the satisfaction of Ubergoals to their expected satisfaction
  - Of course, there are various mathematical functions (e.g. $p'th$ power averages[1] for different p) that one can use to average the satisfaction of multiple Ubergoals; and choices here, i.e.. different specific ways of calculating Pleasure, could lead to systems with different "personalities"
- Goals are Nodes or Links that are on the system's list of goals (the GoalPool). Ubergoals are automatically Goals, but there will be many other Goals also
  - *Example:* The Ubergoal of getting reward from teachers might spawn subgoals like "getting reward from Bob" (if Bob is a teacher), or "making teachers smile" or "create surprising new structures" (if the latter often garners teacher reward). The subgoal of "create surprising new structures" might, in the context of a new person entering the agent's environment with a bag of toys, lead to the creation of a subgoal of asking for a new toy of the sort that could be used to help create new structures. Etc.
- Psi's memory is CogPrime 's AtomTable, with associated structures like the ProcedureRepository (explained in Chapter 19), the SpaceServer and TimeServer (explained in Chapter 26), etc.

[1] the $p'th$ power average is defined as $\sqrt[p]{\sum X^p}$

  - *Examples:* The knowledge of what blocks look like and the knowledge that tall structures often fall down, go in the AtomTable; specific procedures for picking up blocks of different shapes go in the ProcedureRepository; the layout of a room or a pile of blocks at a specific point in time go in the SpaceServer; the series of events involved in the building-up of a tower are temporally indexed in the TimeServer.
  - In Psi and MicroPsi, these same phenomena are stored in memory in a rather different way, yet the basic Psi motivational dynamics are independent of these representational choices

- Psi's "motive selection" process is carried out in CogPrime by economic attention allocation, which allocates ShortTermImportance to Goal nodes
  - *Example:* The flow of importance from "Get reward from teachers" to "get reward from Bob" to "make an interesting structure with blocks" is an instance of what Psi calls "motive selection". No action is being taken yet, but choices are being made regarding what specific goals are going to be used to guide action selection.

- Psi's action selection plays the same role as CogPrime 's action selection, with the clarification that in CogPrime this is a matter of selecting which *procedures* (i.e. schema) to run, rather than which individual actions to execute. However, this notion exists in Psi as well, which accounts for "automatized behaviors" that are similar to CogPrime schemata; the only (minor) difference here is that in CogPrime automatized behaviors are the default case.
  - *Example:* If the goal "make an interesting structure with blocks" has a high STI, then it may be used to motivate choice of a procedure to execute, e.g. a procedure that finds an interesting picture or object seen before and approximates it with blocks, or a procedure that randomly constructs something and then filters it based on interestingness. Once a blocks-structure-building procedure is chosen, this procedure may invoke the execution of sub-procedures such as those involved with picking up and positioning particular blocks.

- Psi's planning is carried out via various learning processes in CogPrime , including PLN plus procedure learning methods like MOSES or hillclimbing
  - *Example:* If the agent has decided to build a blocks structure emulating a pyramid (which it saw in a picture), and it knows how to manipulate and position individual blocks, then it must figure out a procedure for carrying out individual-block actions that will result in production of the pyramid. In this case, a very inexperienced agent might use MOSES or hillclimbing and "guidedly-randomly" fiddle with different construction procedures until it hit on something workable. A slightly more experience agent would use reasoning based

on prior structures it had built, to figure out a rational plan (like: "start with the base, then iteratively pile on layers, each one slightly smaller than the previous.")

- The modulators are system parameters which may be represented by PredicateNodes, and which must be incorporated appropriately in the dynamics of various MindAgents, e.g.
  - *activation* affects action selection. For instance this may be effected by a process that, each cycle, causes a certain amount of STICurrency to pass to schema satisfying certain properties (those involving physical action, or terminating rapidly). The amount of currency passed in this way would be proportional to the *activation*
  - *resolution level* affects perception schema and MindAgents, causing them to expend less effort in processing perceptual data
  - *certainty* affects inference and pattern mining and concept creation processes, causing them to place less emphasis on certainty in guiding their activities, i.e. to be more accepting of uncertain conclusions. To give a single illustrative example: When backward chaining inference is being used to find values for variables, a "fitness target" of the form $strength \times confidence$ is sometimes used; this may be replaced with $strength^p \times confidence^{2-p}$, where *activation* parameter affects the exponent p, so when p tends to 0 confidence is more important, when p tends to 2 strength is more important and when p tends to 1 strength and confidence are equally important.
  - *selection threshold* may be used to effect a process that, each cycle, causes a certain amount of STICurrency (proportional to the selection threshold) to pass to the Goal Atoms that were wealthiest at the previous cycle.

Based on this run-down, Psi and CogPrime may seem very similar, but that's because we have focused here only on the motivation and emotion aspect. Psi uses a very different knowledge representation than CogPrime ; and in the Psi architecture diagram, nearly all of CogPrime is pushed into the role of "background processes that operate in the memory box." According to the theoretical framework underlying CogPrime , the multiple synergetic processes operating in the memory box are actually the crux of general intelligence. But getting the motivation/emotion framework right is also very important, and Psi seems to do an admirable job of that.

22.4 Implementing Emotion Rules Atop Psi's Emotional Dynamics

Human motivations are largely determined by human emotions, which are the result of humanity's evolutionary heritage and embodiment, which are quite different than the heritage and embodiment of current AI systems. So, if we want to create AGI systems that lack humanlike bodies, and didn't evolve to adapt to the same environments as humans did, yet still have vaguely human-like emotional and motivational structures, the latter will need to be explicitly engineered or taught in some way.

For instance, if one wants to make a CogPrime agent display anger, something beyond Psi's model of emotion needs to be coded into the agent to enable this. After all, the rule that when angry the agent has some propensity to harm other beings, is not implicit in Psi and needs to be programmed in. However, making use of Psi's emotion model, anger could be characterized as an emotion consisting of high arousal, low resolution, strong motive dominance, few background checks, strong goal-orientedness (as the Psi model suggests) *and* a propensity to cause harm to agents or objects. This is much simpler than specifying a large set of detailed rules characterizing angry behavior.

The "anger" example brings up the point that desirability of giving AGI systems closely humanlike emotional and motivational systems is questionable. After all we humans cause ourselves a lot of problems with these aspects of our mind/brains, and we sometimes put our more ethical and intellectual sides at war with our emotional and motivational systems. Looking into the future, an AGI with greater power than humans yet a humanlike motivational and emotional system, could be a very dangerous thing.

On the other hand, if an AGI's motivational and emotional system is *too different* from human nature, we might have trouble understanding it, and it understanding us. This problem shouldn't be overblown – it seems possible that an AGI with a more orderly and rational motivational system than the human one might be able to understand us *intellectually* very well, and that we might be able to understand it well using our analytical tools. However, if we want to have mutual *empathy* with an AGI system, then its motivational and emotional framework had better have at least *some* reasonable overlap with our own. The value of empathy for ethical behavior was stressed extensively in Chapter 12 of Part 1.

This is an area where experimentation is going to be key. Our initial plan is to supply CogPrime with rough emulations of some but not all human emotions. We see no need to take explicit pains to simulate emotions like anger, jealousy and hatred. On the other hand, joy, curiosity, sadness, wonder, fear and a variety of other human emotions seem both natural in the context of a robotically or virtually embodied CogPrime system, and valuable in terms of allowing mutual human/CogPrime empathy.

22.4.1 Grounding the Logical Structure of Emotions in the Psi Model

To make this point in a systematic way, we point out that Ortony et al's [OCC90] "cognitive theory of emotions" can be grounded in CogPrime 's version of Psi in a natural way. This theory captures a wide variety of human and animal emotions in a systematic logical framework, so that grounding their framework in CogPrime Psi goes a long way toward explaining how CogPrime Psi accounts for a broad spectrum of human emotions.

The essential idea of the cognitive theory of emotions can be seen in Figure 22.1. What we see there is that common emotions can be defined in terms of a series of choices:

- Is it positive or negative?
- Is it a response to an agent, an event or an object?
- Is it focused on consequences for oneself, or for another?
  - If on another, is it good or bad for the other?
  - If on oneself, is it related to some event whose outcome is uncertain?
    - if it's related to an uncertain outcome, did the expectation regarding the outcome get fulfilled or not?

Figure 22.1 shows how each set of answers to these questions leads to a different emotion. For instance: what is a negative emotion, responding to events, focusing on another, and undesirable to the other? Pity.

In the list of questions, we see that two of them – positive vs. negative, and expectation fulfillment vs. otherwise – are foundational in thee Psi model. The others questions are evaluations that an intelligent agent would naturally make, but aren't bound up with Psi's emotion/motivation infrastructure in such a deep way. Thus, the cognitive theory of emotion emerges as a combination of some basic Psi factors with some more abstract cognitive properties (good vs. bad for another; agents vs. events vs. objects).

22.5 Goals and Contexts

Now we dig deeper into the details of motivation in CogPrime . Just as we have both explicit (local) and implicit (global) memory in CogPrime , we also have both explicit and implicit goals. An explicit goal is formulated as a Goal Atom, and then MindAgents specifically orient the system's activity toward achievement of that goal. An implicit goal is something that the system works toward, but in a more loosely organized way, and without necessarily explicitly representing the knowledge that it is working toward that goal.

Here we will focus mainly on explicit motivation, beginning with a description of Goal Atoms, and the Contexts in which Goals are worked toward via

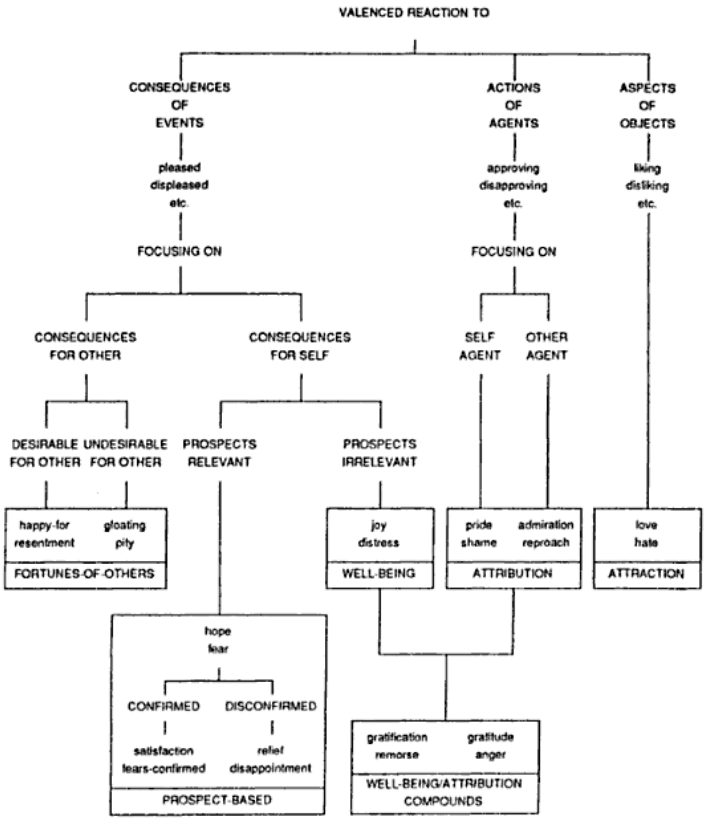


Fig. 22.1 Ontology of Emotions from [OCC90]

executing Procedures. Figure 22.2 gives a rough depiction of the relationship between goals, procedures and context, in a simple example relevant to an OpenCogPrime -controlled virtual agent in a game world.

22.5.1 Goal Atoms

A Goal Atom represents a *target system state* and is true to the extent that the system satisfied the conditions it represents. A Context Atom represents an *observed state of the world/mind,* and is true to the extent that the state it defines is observed. Taken together, these two Atom types provide the infrastructure CogPrime needs to orient its actions in specific contexts toward specific goals. Not all of CogPrime 's activity is guided by these Atoms; much of it is non-goal-directed and spontaneous, or *ambient* as we sometimes call

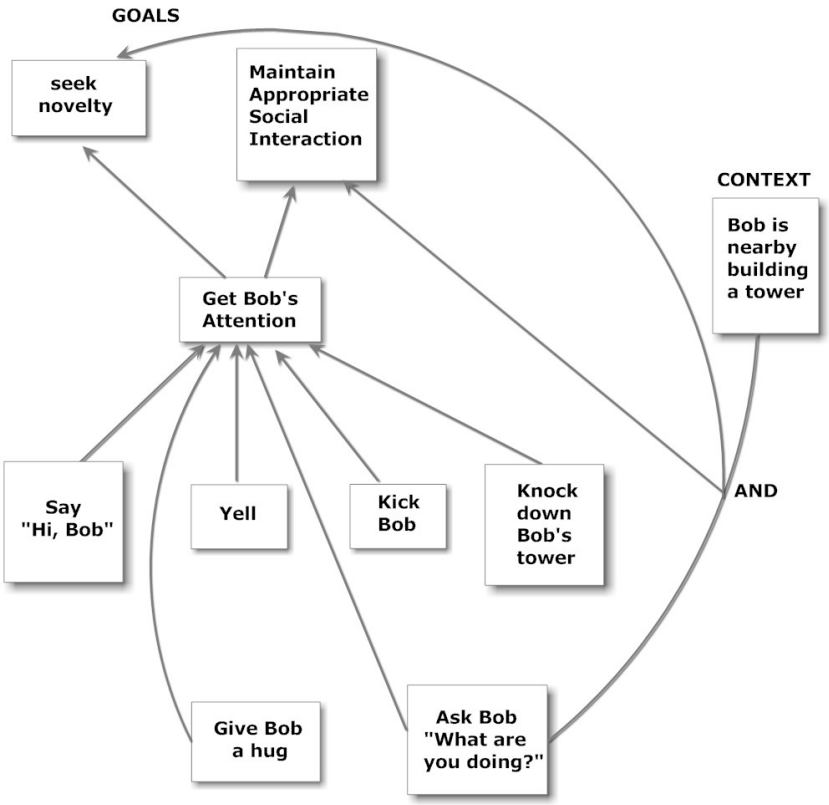


Fig. 22.2 Context, Procedures and Goals. Examples of the basic "goal/context/procedure" triad in a simple game-agent situation.

it. But it is important that some of the system's activity - and in some cases, a substantial portion - is controlled explicitly via goals.

Specifically, a Goal Atom is simply an Atom (usually a PredicateNode, sometimes a Link, and potentially another type of Atom) that has been selected by the GoalRefinement MindAgent as one that represents a state of the atom space which the system finds important to achieve. The extent to which an Atom is considered a Goal Atom at a particular point in time is determined by how much of a certain kind of financial instrument called an RFS (Request For Service) it possesses (as will be explained in Chapter 24).

A CogPrime instance must begin with some initial *Ubergoals* (aka *top level supergoals*), but may then refine these goals in various ways using inference. Immature, "childlike" CogPrime systems cannot modify their Ubergoals nor add nor delete Ubergoals. Advanced CogPrime systems may be allowed to modify, add or delete Ubergoals, but this is a critical and sub-

tle aspect of system dynamics that must be treated with great care. WIKISOURCE:ContextAtom

22.6 Context Atoms

Next, a Context is simply an Atom that is used as the source of a ContextLink, for instance

```
Context
    quantum_computing
    Inheritance Ben amateur
```

or

```
Context
    game_of_fetch
    PredictiveAttraction
        Evaluation give (ball, teacher)
        Satisfaction
```

The former simply says that Ben is an amateur in the context of quantum computing. The latter says that in the context of the game of fetch, giving the ball to the teacher implies satisfaction. A more complex instance pertinent to our running example would be

```
Context
   Evaluation
      Recently
      List
         Minute
         Evaluation
            Ask
            List
               Bob
               ThereExists $X
                  And
                     Evaluation
                        Build
                        List
                           self
                           $X
                     Evaluation
                        surprise
                        List
                           $X
                           Bob
   AverageQuatifier $Y
      PredictiveAttraction
         And
            Evaluation
               Build
               List
```

```
            self
            $Y
         Evaluation
            surprise
            List
               $Y
               Jim
      Satisfaction
```

which says that, if the context is that Bob has recently asked for something surprising to be built, then one strategy for getting satisfaction is to build something that seems likely to satisfy Jim.

An implementation-level note: in the current OpenCogPrime implementation of CogPrime , ContextLinks are implicit rather than explicit entities. An Atom can contain a ComplexTruthValue which in turn contains a number of VersionHandles. Each VersionHandle associates a Context or a Hypothetical with a TruthValue. This accomplishes the same thing as a formal ContextLink, but without the creation of a ContextLink object. However, we continue to use ContextLinks in this book and other documents about CogPrime ; and it's quite possible that future CogPrime implementations might handle them differently.

22.7 Ubergoal Dynamics

In the early phases of a CogPrime system's cognitive development, the goal system dynamics will be quite simple. The Ubergoals are supplied by human programmers, and the system's adaptive cognition is used to derive subgoals. Attentional currency allocated to the Ubergoals is then passed along to the subgoals, as judged appropriate.

As the system becomes more advanced, however, more interesting phenomena may arise regarding Ubergoals: implicit and explicit Ubergoal creation.

22.7.1 Implicit Ubergoal Pool Modification

First of all, *implicit Ubergoal creation or destruction* may occur. Implicit Ubergoal destruction may occur when there are multiple Ubergoals in the system, and some prove easier to achieve than others. The system may then decide not to bother achieving the more difficult Ubergoals. Appropriate parameter settings may militate against this phenomenon, of course.

Implicit Ubergoal creation may occur if some Goal Node G arises that inherits as a subgoal from multiple Ubergoals. This Goal G may then come to act implicitly as an Ubergoal, in that it may get more attentional currency than any of the Ubergoals.

Also, implicit Ubergoal creation may occur via forgetting. Suppose that G becomes a goal via inferred inheritance from one or more Ubergoals. Then, suppose G forgets *why* this inheritance exists, and that in fact the reason becomes obsolete, but the system doesn't realize that and keeps the inheritance there. Then, G is an implicit Ubergoal in a strong sense: it gobbles up a lot of attentional currency, potentially more than any of the actual Ubergoals, but actually doesn't help achieve the Ubergoals, even though the system thinks it does. This kind of dynamic is obviously very bad and should be avoided – and *can* be avoided with appropriate tuning of system parameters (so that the system pays a lot of attention to making sure that its subgoaling-related inferences are correct and are updated in a timely way).

22.7.2 Explicit Ubergoal Pool Modification

An advanced CogPrime system may be given the ability to explicitly modify its Ubergoal pool. This is a very interesting but very subtle type of dynamic, which is not currently well understood and which potentially could lead to dramatically unpredictable behaviors.

However, modification, creation and deletion of goals is a key aspect of human psychology, and the granting of this capability to mature CogPrime systems must be seriously considered.

In the case that Ubergoal pool modification is allowed, one useful heuristic may be to make *implicit Ubergoals* into explicit Ubergoals. For instance: if an Atom is found to consistently receive a lot of RFSs, and has a long time-scale associated with it, then the system should consider making it an Ubergoal. But this heuristic is certainly not sufficient, and any advanced CogPrime system that is going to modify its own Ubergoals should definitely be tuned to put a lot of thought into the process!

The science of Ubergoal pool dynamics basically does not exist at the moment, and one would like to have some nice mathematical models of the process prior to experimenting with it in any intelligent capable CogPrime system. Although Schmiddhuber's Gödel machine [Sch06] has the theoretical capability to modify its ubergoal (note that CogPrime is, in some way, a Gödel machine), there is currently no mathematics allowing us to assess the time and space complexity of such process in a realistic context, given a certain safety confidence target.

22.8 Goal Formation

Goal formation in CogPrime is done via PLN inference. In general, what PLN does for goal formation is to look for predicates that can be proved

to probabilistically imply the existing goals. These new predicates will then tend to receive RFS currency, according to the logic of RFS's to be outlined in Chapter 24, which (according to goal-driven attention allocation dynamics) will make the system more likely to enact procedures that lead to their satisfaction.

As an example of the goal formation process, consider the case where ExternalNovelty is an Ubergoal. The agent may then learn that whenever Bob gives it a picture to look at, its quest for external novelty is satisfied to a singificant degree. That is, it learns

```
Attraction
   Evaluation give (Bob, me, picture)
   ExternalNovelty
```

where *Attraction A B* measures how much A versus $\neg A$ implies B (as explained in Chapter 34). This information allows the agent (the Goal Formation MindAgent) to nominate the atom:

```
EvaluationLink give (Bob, me, picture)
```

as a goal (a subgoal of the original Ubergoal). This is an example of goal refinement, which is one among many ways that PLN can create new goals from existing ones.

22.9 Goal Fulfillment and Predicate Schematization

When there is a Goal Atom G important in the system (with a lot of RFS), the GoalFulfillment MindAgent seeks SchemaNodes S that it has reason to believe, if enacted, will cause G to become true (satisfied). It then adds these to the ActiveSchemaPool, an object to be discussed below. The dynamics by which the GoalFulfillment process works will be discussed in Chapter 24 below.

For example, if a Context Node Chas a high truth value at that time (because it is currently satisfied), and is involved in a relation:

```
Attraction
   C
   PredictiveAttraction S G
```

(for some SchemaNode S and Goal Node G) then this SchemaNode S is likely to be selected by the GoalFulfillment process for execution. This is the fully formalized version of the $Context\&Schema \rightarrow Goal$ notion discussed frequently above. The process may also allow the importance of various schema S to bias its choices of which schemata to execute.

For instance, following up previous examples, we might have

```
Attraction
   Evaluation
```

```
      near
         List
            self
            Bob
   PredictiveAttraction
      Evaluation
         ask
         List
            Bob
            ``Show me a picture''
      ExternalNovelty
```

Of course this is a very simplistic relationship but it's similar to a behavior a young child might display. A more advanced agent would utilize a more abstract relationship that distinguishes various situations in which Bob is nearby, and also involves expressing a concept rather than a particular sentence.

The formation of these schema-context-goal triads may occur according to generic inference mechanisms. However, a specially-focused PredicateSchematization MindAgent is very useful here as a mechanism of inference control, increasing the number of such relations that will exist in the system.

22.10 Context Formation

New contexts are formed by a combination of processes:

- The MapEncapsulation MindAgent, which creates Context Nodes embodying repeated patterns in the perceived world. This process encompasses
  - Maps creating Context Nodes involving Atoms that have high STI at the same time
    - *Example:* A large number of Atoms related to towers could be joined into a single map, which would then be a ConceptNode pointing to "tower-related ideas, procedures and experiences"
  - Maps creating Context Nodes that are involved in a temporal activation pattern that recurs at multiple points in the system's experience.
    - *Example:* There may be a common set of processes involving creating a building out of blocks: first build the base, then the walls, then the roof. This could be encapsulated as a temporal map embodying the overall nature of the process. In this case, the map contains information of the nature: *first do things related to this, then do things related to this, then do things related to this...*
- A set of concept creation MindAgents (see Chapter 38, which fuse and split Context Nodes to create new ones.

- The concept of a building and the concept of a person can be merged to create the concept of a BuildingMan
- The concept of a truck built with Legos can be subdivided into trucks you can actually carry Lego blocks with, versus trucks that are "just for show" and can't really be loaded with objects and then carry them around

22.11 Execution Management

The GoalFulfillment MindAgent chooses schemata that are found likely to achieve current goals, but it doesn't actually execute these schemata. What it does is to take these schemata and place them in a container called the ActiveSchemaPool.

The ActiveSchemaPool contains a set of schemata that have been determined to be reasonably likely, if enacted, to significantly help with achieving the current goal-set. I.e., everything in the active schema pool should be a schema S so that it has been concluded that

```
Attraction
   C
   PredictiveAttraction S G
```

– where C is a currently applicable context and G is one of the goals in the current goal pool – has a high truth value compared to what could be obtained from other known schemata S or other schemata S that could be reasonably expected to be found via reasoning.

The decision of which schemata in the ActiveSchemaPool to enact is made by an object called the ExecutionManager, which is invoked each time the SchemaActivation MindAgent is executed. The ExecutionManager is used to select which schemata to execute, based on doing reasoning and consulting memory regarding which active schemata can usefully be executed simultaneously without causing *destructive interference* (and hopefully causing constructive interference). This process will also sometimes (indirectly) cause new schemata to be created and/or other schemata from the AtomTable to be made active. This process is described more fully in Chapter 24 on action selection. WIKISOURCE:GoalsAndTime

For instance, if the agent is involved in building a blocks structure intended to surprise or please Bob, then it might simultaneously carry out some blocks-manipulation schema, and also a schema involving looking at Bob to garner his approval. If it can do the blocks manipulation without constantly looking at the blocks, this should be unproblematic for the agent.

22.12 Goals and Time

The CogPrime system maintains an explicit list of "Ubergoals", which as will be explained in Chapter 24, receive attentional currency which they may then allocate to their subgoals according to a particular mechanism.

However, there is one subtle factor involved in the definition of the Ubergoals: time. The truth value of a Ubergoal is typically defined as the average level of satisfaction of some Demand over some period of time – but the time scale of this averaging can be very important. In many cases, it may be worthwhile to have separate Ubergoals corresponding to the same Demand but doing their truth-value time-averaging over different time scales. For instance, corresponding to Demands such as Novelty or Health, we may posit both long-term and short-term versions, leading to Ubergoals such as CurrentNovelty, LongTermNovelty, CurrentHealth, LongTermHealth, etc. Of course, one could also wrap multiple Ubergoals corresponding to a single Demand into a single Ubergoal combining estimates over multiple time scales; this is not a critical issue and the only point of splitting Demands into multiple Ubergoals is that it can make things slightly simpler for other cognitive processes.

For instance, if the agent has a goal of pleasing Bob, and it knows Bob likes to be presented with surprising structures and ideas, then the agent has some tricky choices to make. Among other choices it must balance between focusing on

- creating things and then showing them to Bob
- studying basic knowledge and improving its skills.

Perhaps studying basic knowledge and skills will give it a foundation to surprise Bob much more dramatically in the mid-term future ... but in the short run will not allow it to surprise Bob much at all, because Bob already knows all the basic material. This is essentially a variant of the general "exploration versus exploitation" dichotomy, which lacks any easy solution. Young children are typically poor at carrying out this kind of balancing act, and tend to focus overly much on near-term satisfaction. There are also significant cultural differences in the heuristics with which adult humans face these issues; e.g. in some contexts Oriental cultures tend to focus more on mid to long term satisfaction whereas Western cultures are more short term oriented.

Chapter 23
Attention Allocation

Co-authored with Joel Pitt and Matt Ikle' and Rui Liu

23.1 Introduction

The critical factor shaping real-world general intelligence is resource constraint. Without this issue, we could just have simplistic program-space-search algorithms like AIXI^{tl} instead of complicated systems like the human brain or CogPrime . Resource constraint is managed implicitly within various components of CogPrime , for instance in the population size used in evolutionary learning algorithms, and the depth of forward or backward chaining inference trees in PLN. But there is also a component of CogPrime that manages resources on a global and cognitive-process-independent manner: the *attention allocation* component.

The general principles the attention allocation process should follow are easy enough to see: History should be used as a guide, and an intelligence should make probabilistic judgments based on its experience, guessing which resource-allocation decisions are likely to maximize its goal-achievement. The problem is that this is a difficult learning and inference problem, and to carry it out with excellent accuracy would require a limited-resources intelligent system to spend nearly all its resources deciding what to pay attention to and nearly none of them actually paying attention to anything else. Clearly this would be a very poor allocation of an AI system's attention! So simple heuristics are called for, to be supplemented by more advanced and expensive procedures on those occasions where time is available and correct decisions are particularly crucial.

Attention allocation plays, to a large extent, a "meta" role in enabling mind-world correspondence. Without effective attention allocation, the other cognitive processes can't do their jobs of helping an intelligent agent to achieve its goals in an environment, because they won't be able to pay attention to the most important parts of the environment, and won't get computational resources at the times when they need it. Of course this need could be addressed in multiple different ways. For example, in a system with multi-

ple complex cognitive processes, one could have attention allocation handled separately within each cognitive process, and then a simple "top layer" of attention allocation managing the resources allocated to each cognitive process. On the other hand, one could also do attention allocation via a single dynamic, pervasive both within and between individual cognitive processes. The CogPrime design gravitates more toward the latter approach, though also with some specific mechanisms within various MindAgents; and efforts have been made to have these specific mechanisms modulated by the generic attention allocation structures and dynamics wherever possible.

In this chapter we will dig into the specifics of how these *attention allocation* issues are addressed in the CogPrime design. In short, they are addressed via a set of mechanisms and equations for dynamically adjusting *importance values* attached to Atoms and MindAgents. Different importance values exist pertinent to different time scales, most critically the short-term (STI) and long-term (LTI) importances. The use of two separate time-scales here reflects fundamental aspects of human-like general intelligence and real-world computational constraints.

The dynamics of STI is oriented partly toward the need for real-time responsiveness, and the more thoroughgoing need for cognitive processes at speeds vaguely resembling the speed of "real time" social interaction. The dynamics of LTI is based on the fact that some data tends to be useful over long periods of time, years or decades in the case of human life, but the practical capability to store large amounts of data in a rapidly accessible way is limited. One could imagine environments in which very-long-term multiple-type memory was less critical than it is in typical human-friendly environments; and one could envision AGI systems carrying out tasks in which real-time responsiveness was unnecessary (though even then some attention focusing would certainly be necessary). For AGI systems like these, an attention allocation system based on STI and LTI with CogPrime -like equations would likely be inappropriate. But for an AGI system intended to control a vaguely human-like agent in an environment vaguely resembling everyday human environments, the focus on STI and LTI values, and the dynamics proposed for these values in CogPrime , appear to make sense.

Two basic innovations are involved in the mechanisms attached to these STI and LTI importance values:

- treating attention allocation as a data mining problem: the system records information about what it's done in the past and what goals it's achieved in the past, and then recognizes patterns in this history and uses them to guide its future actions via probabilistically adjusting the (often context-specific) *importance values* associated with internal terms, actors and relationships, and adjusting the "effort estimates" associated with Tasks
- using an artificial-economics approach to update the importance values (attached to Atoms, MindAgents, and other actors in the CogPrime system) that regulate system attention. (And, more speculatively, using an

information geometry based approach to execute the optimization involved in the artificial economics approach efficiently and accurately.)

The integration of these two aspects is crucial. The former aspect provides fundamental data about what's of value to the system, and the latter aspect allows this fundamental data to be leveraged to make sophisticated and integrative judgments rapidly. The need for the latter, rapid-updating aspect exists partly because of the need for real-time responsiveness, imposed by the need to control a body in a rapidly dynamic world, and the prominence in the architecture of an animal-like cognitive cycle. The need for the former, data-mining aspect (or something functionally equivalent) exists because, in the context of the tasks involved in human-level general intelligence, the assignment of credit problem is hard – the relations between various entities in the mind and the mind's goals are complex, and identifying and deploying these relationships is a difficult learning problem requiring application of sophisticated intelligence.

Both of these aspects of attention allocation dynamics may be used in computationally lightweight or computationally sophisticated manners:

- For routine use in real-time activity
  - "data mining" consists of forming HebbianLinks (involved in the associative memory and inference control, see Section 23.5), where the weight of the link from Atom A to Atom B is based on the probability of shared utility of A and B
  - economic attention allocation consists of spreading ShortTermImportance and LongTermImportance "artificial currency" values (both grounded in the universal underlying "juju" currency value defined further below) between Atoms according to specific equations that somewhat resemble neural net activation equations but respect the conservation of currency
- For use in cases where large amounts of computational resources are at stake based on localized decisions, hence allocation of substantial resources to specific instances of attention-allocation is warranted
  - "data mining" may be more sophisticated, including use of PLN, MOSES and pattern mining to recognize patterns regarding what probably deserves more attention in what contexts
  - economic attention allocation may involve more sophisticated economic calculations involving the expected future values of various "expenditures" of resources

The particular sort of "data mining" going on here is definitely not exactly what the human brain does, but we believe this is a case where slavish adherence to neuroscience would be badly suboptimal (even if the relevant neuroscience were well known, which is not the case). Doing attention allocation entirely in a distributed, formal-neural-net-like way is, we believe, extremely

and unnecessarily inefficient, and given realistic resource constraints it leads to the rather poor attention allocation that we experience every day in our ordinary waking state of consciousness. Several aspects of attention allocation can be fruitfully done in a distributed, neural-net-like way, but not having a logically centralized repository of system-history information (regardless of whether it's physically distributed or not) seems intrinsically problematic in terms of effective attention allocation. And we argue that, even for those aspects of attention allocation that are best addressed in terms of distributed, vaguely neural-net-like dynamics, an artificial-economics approach has significant advantages over a more strictly neural-net-like approach, due to the greater ease of integration with other cognitive mechanisms such as forgetting and data mining.

23.2 Semantics of Short and Long Term Importance

We now specify the two types of importance value (short and long term) that play a key role in CogPrime dynamics. Conceptually, ShortTermImportance (STI) is defined as

$$STI(A) = P(A \text{ will be useful in the near future})$$

whereas LongTermImportance (LTI) is defined as

$$LTI(A) = P(A \text{ will be useful eventually, in the foreseeable future})$$

Given a time-scale T, in general we can define an importance value relative to T as

$$I_T(A) = P(A \text{ will be useful during the next } T \text{ seconds})$$

In the ECAN module in CogPrime , we deal only with STI and LTI rather than any other importance values, and the dynamics of STI and LTI are dealt with by treating them as two separate "artificial currency" values, which however are interconvertible via being mutually grounded in a common currency called "juju."

For instance, if the agent is intensively concerned with trying to build interesting blocks structures, then knowledge about interpreting biology research paper abstracts is likely to be of very little current importance. So its biological knowledge will get low STI, but – assuming the agent expects to use biology again – it should maintain reasonably high LTI so it can remain in memory for future use. And if in its brainstorming about what blocks structures to build, the system decides to use some biological diagrams as inspiration, STI can always spread to some of the biology-related Atoms, increasing their relevance and getting them more attention. While the attention allocation system contains mechanisms to convert STI to LTI, it also has

parameter settings biasing it to spend its juju on both kinds of importance – i.e. it contains an innate bias to both focus its attention judiciously, and manage its long-term memory conscientiously.

Because in CogPrime most computations involving STI and LTI are required to be very rapid (as they're done for many Atoms in the memory very frequently), in most cases when dealing with these quantities, it will be appropriate to sacrifice accuracy for efficiency. On the other hand, it's useful to *occasionally* be able to carry out expensive, highly accurate computations involving importance.

An example where doing expensive computations about attention allocation might pay off, would be the decision whether to use biology-related or engineering-related metaphors in creating blocks structures to please a certain person. In this case it could be worth doing a few steps of inference to figure out whether there's a greater intensional similarity between that person's interests and biology or engineering; and then using the results to adjust the STI levels of whichever of the two comes out most similar. This would not be a particularly expensive inference to carry out, but it's still much more effort than what can be expended on Atoms in the memory most of the time. Most attention allocation in CogPrime involves simple neural-net type spreading dynamics rather than explicit reasoning.

Figure 23.1 illustrates the key role of LTI in the forgetting process. Figure 23.2 illustrates the key role of STI in maintaining a "moving bubble of attention", which we call the system's AttentionalFocus.

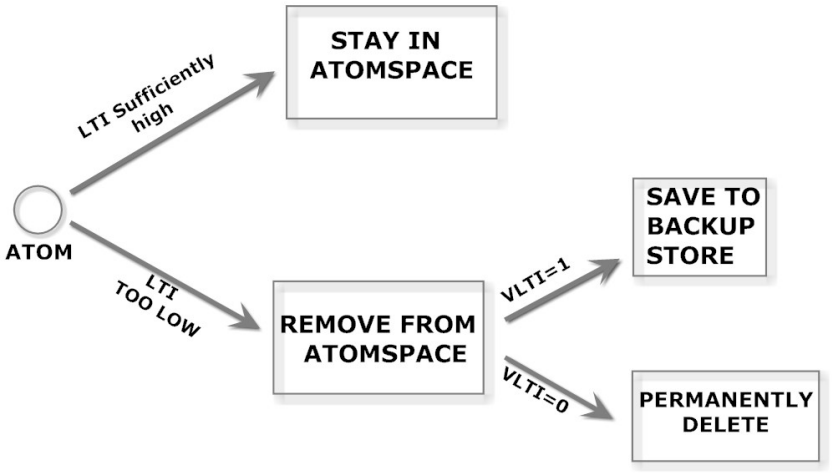


Fig. 23.1 LongTermImportance and Forgetting.

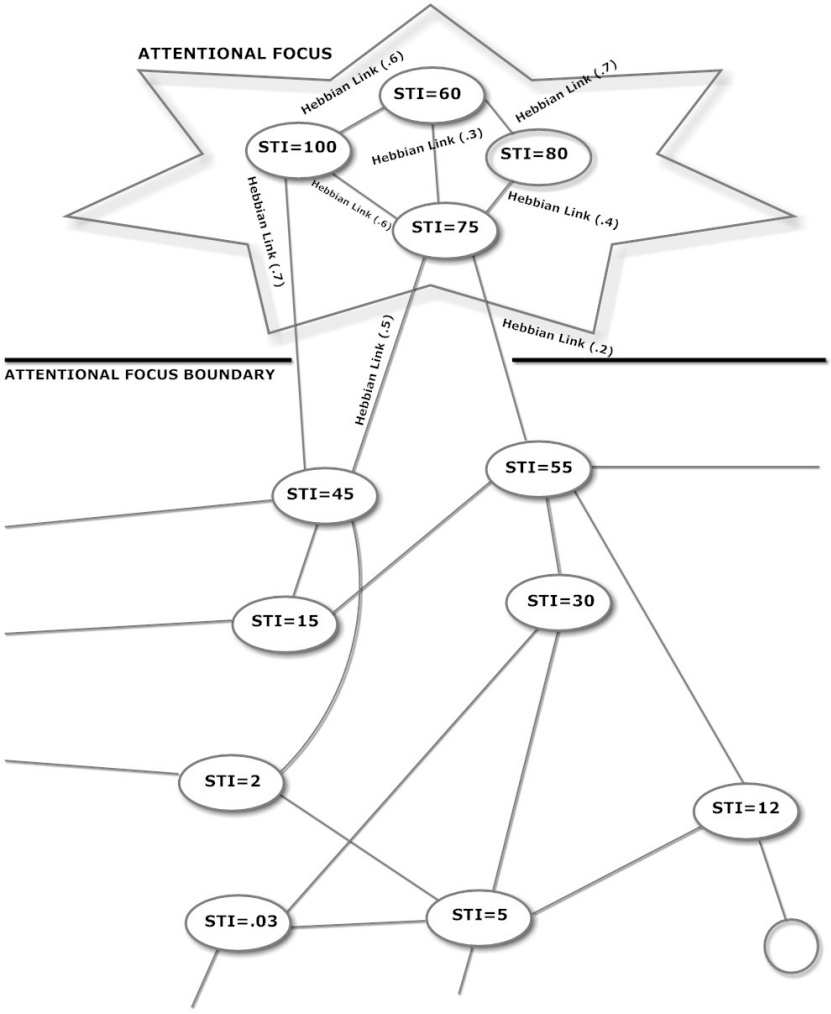


Fig. 23.2 Formation of the AttentionalFocus. The dynamics of STI is configured to encourage the emergence of richly cross-connected networks of Atoms with high STI (above a threshold called the AttentionalFocusBoundary), passing STI among each other as long as this is useful and forming new HebbianLinks among each other. The collection of these Atoms is called the AttentionalFocus.

23.2.1 The Precise Semantics of STI and LTI

Now we precisiate the above definitions of STI and LTI.

First, we introduce the notion of *reward*. Reward is something that Goals give to Atoms. In principle a Goal might give an Atom reward in various different forms, though in the design given here, reward will be given in

units of a currency called *juju*. The process by which Goals assign reward to Atoms is part of the "assignment of credit" process (and we will later discuss the various time-scales on which assignment of credit may occur and their relationship to the time-scale parameter within LTI).

Next, we define

$J(A, t_1, t_2, r)$ = expected amount of reward A will receive between t_1 and t_2 time-steps in the future, if its STI has percentile rank r among all Atoms in the AtomTable

The percentile rank r of an Atom is the rank of that Atom in a list of Atoms ordered by decreasing STI, divided by the total number of Atoms. The reason for using a percentile rank instead of the STI itself is because at any given time only a limited number of atoms can be given attention, so all atoms below a certain perceptible rank, depending on the amount of available resource, will simply be ignored.

This is a fine-grained measure of how worthwhile it is expected to be to increase A's STI, in terms of getting A rewarded by Goals.

For practical purposes it is useful to collapse $J(A, t_1, t_2, r)$ to a single number:

$$J(A, t_1, t_2) = \frac{\sum_r J(A, t_1, t_2, r) w_r}{\sum_r w_r}$$

where w_r weights the different percentile ranks (and should be chosen to be monotone increasing in r). This is a single-number measure of the responsiveness of an Atom's utility to its STI level. So for instance if A has a lot of STI and it turns out to be rewarded then J(A, t1, t2) will be high. On the other hand if A has little STI then whether it gets rewarded or not will not influence J(A, t1, t2) much.

To simplify notation, it's also useful to define a single-time-point version

$$J(A, t) = J(A, t, t)$$

23.2.1.1 Formalizing STI

Using these definitions, one simple way to make the STI definition precise is:

$$STI_{thresh}(A, t) = P(J(A, t, t + t_{short}) \geq s_{threshold})$$

where $s_{threshold}$ demarcates the "attentional focus boundary." Which is a way of saying that we don't want to give STI to atoms that would not get rewarded if they were given attention.

Or one could make the STI definition precise in a fuzzier way, and define

$$STI_{fuzzy}(A,t) = \sum_{s=0}^{\infty} J(A,t+s)c^{-s}$$

for some appropriate parameter c (or something similar with a decay function less severe than exponential)

In either case, the goal of the ECAN subsystem, regarding STI, is to assign each Atom A an STI value that corresponds as closely as possible to the theoretical STI values defined by whichever one of the above equations is selected (or some other similar equation).

23.2.2 STI, STIFund, and Juju

But how can one estimate these probabilities in practice? In some cases they may be estimated via explicit inference. But often they must be estimated by heuristics.

The estimative approach taken in current CogPrime design is an artificial economy, in which each Atom maintains a certain fund of artificial currency. In the current proposal this currency is called juju and is the same currency used to value LTI. Let us call the amount of juju owned by Atom A the STIFund of A. Then, one way to formalize the goal of the artificial economy is to state that: if one ranks all Atoms by the wealth of their STIFund, and separately ranks all Atoms by their theoretical STI value, the rankings should be as close as possible to the same. One may also formalize the goal in terms of value correlation instead of rank correlation, of course.

Proving conditions under which the STIFund values will actually correlate well with the theoretical STI values, is an open math problem. Heuristically, one may map STIFund values into theoretical STI values by a mapping such as

$$A.STI = \alpha + \beta \frac{A.STIFund - STIFund.\min}{STIFund.\max - STIFund.\min}$$

where $STIFund.\min = \min_X X.STIFund$. However, we don't currently have rigorous grounding for any particular functional form for such a mapping; the above is just a heuristic approximation.

The artificial economy approach leads to a variety of supporting heuristics. For instance, one such heuristic is: if A has been used at time t, then it will probably be useful at time $t+s$ for small s. Based on this heuristic, whenever a MindAgent uses an Atom A, it may wish increase A's STIFund (so as to hopefully increase correlation of A's STIFund with its theoretical STI). It does so by transferring some of its juju to A's STIFund.

23.2.3 Formalizing LTI

Similarly to with STI, with LTI we will define theoretical LTI values, and posit an LTIFund associated with each Atom, which seeks to create values correlated with the theoretical LTI values.

For LTI, the theoretical issues are subtler. There is a variety of different ways to precisiate the above loose conceptual definition of LTI. For instance, one can (and we will below) create formalizations of both:

1. $LTI_{cont}(A)$ = (some time-weighting or normalization of) the expected value of A's total usefulness over the long-term future
2. $LTI_{burst}(A)$ = the probability that A ever becomes *highly useful* at some point in the long-term future

(here "cont" stands for "continuous"). Each of these may be formalized, in similar but nonidentical ways.

These two forms of LTI may be viewed as extremes along a continuum; one could posit a host of intermediary LTI values between them. For instance, one could define

$LTI_p(A)$ = the $p'th$ power average[1] of expectation of the utility of A over brief time intervals, measured over the long-term future

Then we would have

$$LTI_{burst} = LTI_{\infty}$$

$$LTI_{cont} = LTI_1$$

and could vary p to vary the sharpness of the LTI computation. This might be useful in some contexts, but our guess is that it's overkill in practice and that looking at LTI_{burst} and LTI_{cont} is enough (or more than enough; the current OCP code uses only one LTI value and that has not been problematic so far).

23.2.4 Applications of LTI_{burst} versus LTI_{cont}

It seems that the two forms of LTI discussed above might be of interest in different contexts, depending on the different ways that Atoms may be used so as to achieve reward.

If an Atom is expected to get rewarded for the results of its being selected by MindAgents that carry out diffuse, background thinking (and hence often select low-STI Atoms from the AtomTable), then it may be best associated with LTI_{cont}.

[1] the $p'th$ power average is defined as $\sqrt[p]{\sum X^p}$

On the other hand, if an Atom is expected to get rewarded for the results of its being selected by MindAgents that are focused on intensive foreground thinking (and hence generally only select Atoms with very high STI), it may be best associated with LTI_{burst}.

In principle, Atoms could be associated with particular LTI_p based on the particulars of the selection mechanisms of the MindAgents expected to lead to their reward. But the issue with this is, it would result in Atoms carrying around an excessive abundance of different LTI_p values for various p, resulting in memory bloat; and it would also require complicated analyses of MindAgent dynamics. If we do need more than one LTI value, one would hope that two will be enough, for memory conservation reasons.

And of course, if an Atom has only one LTI value associated with it, this can reasonably be taken to stand in for the other one: either of LTI_{burst} or LTI_{cont} may, in the absence of information to the contrary, be taken as an estimate of the other.

23.2.4.1 LTI with Various Time Lags

The issue of the p value in the average in the definition of LTI is somewhat similar to (though orthogonal to) the point that there are many different interpretations of LTI, achieved via considering various time-lags. Our guess is that a small set of time-lags will be sufficient. Perhaps one wants an exponentially increasing series of time-lags: i.e. to calculate LTI over k cycles where k is drawn from $\{r, 2r, 4r, 8r, ...2^N r\}$.

The time-lag in LTI seems related to the time-lag in the system's goals. If a Goal object is disseminating juju, and the Goal has an intrinsic time scale of t, then it may be interested in LTI on time-scale t. So when a MA (MindAgent) is acting in pursuit of that goal, it should spend a bunch of its juju on LTI on time-scale t.

Complex goals may be interested in multiple time-scales (for instance, a goal might place greater value on things that occur in the next hour, but still have nonzero interest in things that occur in a week), and hence may have different levels of interest in LTI on multiple time-scales.

23.2.4.2 Formalizing Burst LTI

Regarding burst LTI, two approaches to formalization seem to be the threshold version

$$LTI_{burst,thresh}(A) = P(A \text{ will receive a total of at least } s_{threshold} \text{ amount of normalized stimulus during some time interval of length } t_{short} \text{ in the next } t_{long} \text{ time steps})$$

and the fuzzy version,

$$LTI_{burst,fuzzy}(A,t) = \sum_{s=0}^{\infty} J(A, t+s, t+s+t_{short}) f(s, t_{long})$$

where $f(t, t_{long}) : R^+ \times R^+ \to R^+$ is a nonincreasing function that remains roughly constant in t up till a point t_{long} steps in the future, and then begins slowly decaying.

23.2.4.3 Formalizing Continuous LTI

The threshold version of continuous LTI is quite simply:

$$LTI_{cont,thresh}(A, t_{long}) = STI_{thresh}(A, t_{long})$$

That is, smooth threshold LTI is just like smooth threshold STI, but the time-scale involved is longer.

On the other hand, the fuzzy version of smooth LTI is:

$$LTI_{cont,fuzzy}(A,t) = \sum_{s=0}^{\infty} J(A, t+s) f(s, t_{long})$$

using the same decay function f that was introduced above in the context of burst LTI.

23.3 Defining Burst LTI in Terms of STI

It is straightforward to define burst LTI in terms of STI, rather than directly in terms of juju. We have

$$LTI_{burst,thresh}(A,t) = P(\bigcup_{s=0}^{s=t_{long}} STI_{thresh}(A, t+s))$$

Or, using the fuzzy definitions, we obtain instead the *approximate* equation

$$LTI_{burst,fuzzy}(A,t) \approx \sum_{s=0}^{\infty} \alpha(s) STI_{fuzzy}(A, t+s) f(s, t_{long})$$

where

$$\alpha(s) = \frac{1-c}{1-c^{s+1}}$$

or the more complex exact equation:

$$LTI_{burst,fuzzy}(A,t) = \sum_{s=0}^{\infty} STI_{fuzzy}(A, t+s) \left(f(s, t_{long}) - \sum_{r=1}^{s} (c^{-r} f(s-r, t_{long})) \right)$$

23.4 Valuing LTI and STI in terms of a Single Currency

We now further discuss the approach of defining LTIFund and STIFund in terms of a single currency: juju (which as noted, corresponds in the current ECAN design to normalized stimulus).

In essence, we can think of STIFund and LTIFund as different forms of financial instrument, which are both grounded in juju. Each Atom has two financial instruments attached to it: "STIFund of Atom A" and "LTIFund of Atom A" (or more if multiple versions of LTI are used). These financial instruments have the peculiarity that, although many agents can put juju into any one of them, no record is kept of who put juju in which one. Rather, the MA's are acting so as to satisfy the system's Goals, and are adjusting the STIFund and LTIFund values in a heuristic manner that is expected to approximately maximize the total utility propagated from Goals to Atoms.

Finally, each of these financial instruments has a value that gets updated by a specific update equation.

To understand the logic of this situation better, consider the point of view of a Goal with a certain amount of resources (juju, to be used as reward), and a certain time-scale on which its satisfaction is to be measured. Suppose that the goal has a certain amount of juju to expend on getting itself satisfied.

This Goal clearly should allocate some of its juju toward getting processor time allocated toward the right Atoms to serve its ends in the near future; and some of its juju toward ensuring that, in future, the memory will contain the Atoms it will want to see processor time allocated to. Thus, it should allocate some of its juju toward boosting the STIFund of Atoms that it thinks will (if chosen by appropriate MindAgents) serve its needs in the near future, and some of its juju toward boosting the LTIFund of Atoms that it thinks will serve its need in the future (if they remain in RAM). Thus, when a Goal invokes a MindAgent (giving the MindAgent the juju it needs to access Atoms and carry out its work), it should tell this MindAgent to put some of its juju into LTIFunds and some into STIFunds.

If a MindAgent receives a certain amount of juju each cycle, independently of what the system Goals are explicitly telling it, then this should be viewed as reflecting an implicit goal of "ambient cognition", and the balance of STI and LTI associated with this implicit goal must be a system parameter.

In general, the trade-off between STI and LTI boils down to the weighting between near and far future that is intrinsic to a particular Goal. Simplistically: if a Goal values getting processor allocated to the right stuff *immediately* 25 times more than getting processor allocated to the right stuff $20K$ cycles in the future, then it should be willing spend $25\times$ more of its juju on STI than on $LTI_{20K\ cycles}$. (This simplistic picture is complicated a little by the relationship between different time-scales. For instance, boosting $LTI_{10K\ cycles}(A)$ will have an indirect effect of increasing the odds that A will still be in memory $20K$ cycles in the future.)

However, this isn't the whole story, because multiple Goals are setting the importance values of the same set of Atoms. If M_1 pumps all its juju into STI for certain Atoms, then M_2 may decide it's not worthwhile for it to bother competing with M_1 in the STI domain, and to spend its juju on LTI instead.

Note that the current system doesn't allow a MA to change its mind about LTI allocations. One can envision a system where a MindAgent could in January pay juju to have Atom A kept around for a year, but then change its mind in June 6 months later, and ask for some of the money back. But this would require an expensive accounting procedure, keeping track of how much of each Atom's LTI had been purchased by which MindAgent; so it seems a poor approach.

A more interesting alternative would be to allow MA's to retain adjustable "reserve funds" of juju. This would mean that a MindAgent would never see a purpose to setting $LTI_{one\ year}(A)$ instead of repeatedly setting $LTI_{one\ minute}$, unless a substantial transaction cost were incurred with each transaction of adjusting an Atom's LTI. Introducing a transaction cost plus an adjustable per-MindAgent juju reserve fund, and LTI's on multiple time scales, would give the LTI framework considerable flexibility. (To prevent MA's from hoarding their juju, one could place a tax rate on reserve juju.)

The conversion rate between STI and LTI becomes an interesting matter; though it seems not a critical one, since in the practical dynamics of the system it's juju that is used to produce STI and LTI. In the current design there is no apparent reason to spread STI of one Atom to LTI of another Atom, or convert the STI of an Atom into LTI of that same Atom, etc. – but such an application might come up. (For the rest of this paragraph, let's just consider LTI with one time scale, for simplicity.) Each Goal will have its own preferred conversion rate between STI and LTI, based on its own balancing of different time scales. But, each Goal will also have a limited amount of juju, hence one can only trade a certain amount of STI for LTI, if one is trading with a specific goal G. One could envision a centralized STI-for-LTI market where different MA's would trade with each other, but this seems overcomplicated, at least at the present stage.

As a simpler software design point, this all suggests a value for associating each Goal with a parameter telling how much of its juju it wants to spend on STI versus LTI. Or, more subtly, how much of its juju it wants to spend on LTI on various time-scales. On the other hand, in a simple ECAN implementation this balance may be assumed constant across all Goals.

23.5 Economic Attention Networks

Economic Attention Networks (ECANs) are dynamical systems based on the propagation of STI and LTI values. They are similar in many respects to Hopfield nets, but are based on a different conceptual foundation involving

the propagation of amounts of (conserved) currency rather than neural-net activation. Further, ECANs are specifically designed for integration with a diverse body of cognitive processes as embodied in integrative AI designs such as CogPrime . A key aspect of the CogPrime design is the imposition of ECAN structure on the CogPrime AtomSpace.

Specifically, ECANs have been designed to serve two main purposes within CogPrime : to serve as an associative memory for the network, and to facilitate effective allocation of the attention of other cognitive processes to appropriate knowledge items.

An ECAN is simply a graph, consisting of un-typed nodes and links, and also "Hebbian" links that may have types such as HebbianLink, InverseHebbianLink, or SymmetricHebbianLink. Each node and link in an ECAN is weighted with two currency values, called STI (short-term importance) and LTI (long-term importance); and each Hebbian link is weighted with a probabilistic truth value.

The equations of an ECAN explain how the STI, LTI and Hebbian link weights values get updated over time. As alluded to above, the metaphor underlying these equations is the interpretation of STI and LTI values as (separate) artificial currencies. The fact that STI and LTI are currencies means that, except in unusual instances where the ECAN controller decides to introduce inflation or deflation and explicitly manipulate the amount of currency in circulation, the total amounts of STI and LTI in the system are conserved. This fact makes the dynamics of an ECAN dramatically different than that of an attractor neural network.

In addition to STI and LTI as defined above, the ECAN equations also contain the notion of an Attentional Focus (AF), consisting of those Atoms in the ECAN with the highest STI values (and represented by the $s_{threshold}$ value in the above equations). These Atoms play a privileged role in the system and, as such, are treated using an alternate set of equations.

23.5.1 Semantics of Hebbian Links

Conceptually, the probability value of a HebbianLink from A to B is the odds that if A is in the AF, so is B; and correspondingly, the InverseHebbianLink from A to B is weighted with the odds that if A is in the AF, then B is not. An ECAN will often be coupled with a "Forgetting" process that removes low-LTI Atoms from memory according to certain heuristics. A critical aspect of the ECAN equations is that Atoms periodically spread their STI and LTI to other Atoms that connect to them via Hebbian and InverseHebbianLinks; this is the ECAN analogue of activation spreading in neural networks.

Multiple varieties of HebbianLink may be constructed, for instance

- Asymmetric HebbianLinks, whose semantics are as mentioned above: the truth value of *HebbianLink A B* denotes the probability that if A is in the AF, so is B
- Symmetric HebbianLinks, whose semantics are that: the truth value of *SymmetricHebbianLink A B* denotes the probability that if one of A or B is in the AF, both are

It is also worth noting that one can combine ContextLinks with HebbianLinks and express contextual association such that in context C, there is a strong HebbianLink between A and B.

23.5.2 Explicit and Implicit Hebbian Relations

In addition to explicit HebbianLinks, it can be useful to treat other links implicitly as HebbianLinks. For instance, if ConceptNodes A and B are found to connote similar concepts, and a SimilarityLink is formed between them, then this gives reason to believe that maybe a SymmetricHebbianLink between A and B should exist as well. One could incorporate this insight in CogPrime in at least three ways:

- creating HebbianLinks paralleling other links (such as SimilarityLinks)
- adding "Hebbian weights" to other links (such as SimilarityLinks)
- implicitly interpreting other links (such as SimilarityLinks) as HebbianLinks

Further, these strategies may potentially be used together.

There are some obvious semantic relationships to be used in interpreting other link types implicitly as HebbianLinks: for instance, Similarity maps into SymmetricHebbian, and *Inheritance A B* maps into *Hebbian A B*. One may express these as inference rules, e.g.

```
SimilarityLink A B <tv_1>
|-
SymmetricHebbianLink A B <tv_2>
```

where $tv_2.s = tv_1.s$. Clearly, $tv_2.c < tv_1.c$; but the precise magnitude of $tv2.c$ must be determined by a heuristic formula. One option is to set $tv_2.c = \alpha tv_1.c$ where the constant α is set empirically via data mining the System Activity Tables to be described below.

23.6 Dynamics of STI and LTI Propagation

We now get more specific about how some of these ideas are implemented in the currently implemented ECAN subsystem of CogPrime Prime. We'll

discuss mostly STI here because in the current design LTI works basically the same way.

MindAgents send out stimulus to Atoms whenever they use them (or else, sometimes, just for the purpose of increasing the Atom's STI); and before these stimulus values are used to update the STI levels of the receiving Atom, they are normalized by: the total amount of stimulus sent out by the MindAgent in that cycle, multiplied by the total amount of STI currency that the MindAgent decided to spend in that cycle. The normalized stimulus is what has above been called juju. This normalization preserves fairness among MA's, and conservation of currency.

(The reason "stimuli" exist, separately from STI, is that stimulus-sending needs to be very computationally cheap, as in general it's done frequently by each MA each cycle, and we don't want each action a MA takes to invoke some costly importance-updating computation.)

Then, Atoms exchange STI according to certain equations (related to HebbianLinks and other links), and have their STI values updated according to certain equations (which involve, among other operations, transferring STI to the "central bank").

23.6.1 ECAN Update Equations

The CogServer is understood to maintain a kind of central bank of STI and LTI funds. When a non-ECAN MindAgent finds an Atom valuable, it sends that Atom a certain amount of Stimulus, which results in that Atom's STI and LTI values being increased (via equations to be presented below, that transfer STI and LTI funds from the CogServer to the Atoms in question). Then, the ECAN ImportanceUpdating MindAgent carries out multiple operations, including some that transfer STI and LTI funds from some Atoms back to the CogServer.

There are multiple ways to embody this process equationally; here we briefly describe two variants.

23.6.1.1 Definition and Analysis of Variant 1

We now define a specific set of equations in accordance with the ECAN conceptual framework described above. We define $H_{STI} = [s_1, \cdots, s_n]$ to be the vector of STI values, and $C = \begin{bmatrix} c_{11}, \cdots, c_{1n} \\ \vdots \ddots \vdots \\ c_{n1}, \cdots, c_{nn} \end{bmatrix}$ to be the connection matrix of Hebbian probability values, where it is assumed that the existence of a HebbianLink or InverseHebbianLink between A and B are mutually exclu-

sive possibilities. We also define $C_{LTI} = \begin{bmatrix} g_{11}, \cdots, g_{1n} \\ \vdots \ddots \vdots \\ g_{n1}, \cdots, g_{nn} \end{bmatrix}$ to be the matrix of LTI values for each of the corresponding links.

We assume an updating scheme in which, periodically, a number of Atoms are allocated Stimulus amounts, which causes the corresponding STI values to change according to the equations

$$\forall i : s_i = s_i - \text{rent} + \text{wages},$$

where rent and wages are given by

$$\text{rent} = \begin{cases} \langle \text{Rent} \rangle \cdot \max\left(0, \frac{\log\left(\frac{20 s_i}{\text{recentMaxSTI}}\right)}{2}\right), & \text{if } \ s_i > 0 \\ 0, & \text{if } \ s_i \leq 0 \end{cases}$$

and

$$\text{wages} = \begin{cases} \frac{\langle \text{Wage} \rangle \langle \text{Stimulus} \rangle}{\sum_{i=1}^{n} p_i}, \text{if } \ p_i = 1 \\ \frac{\langle \text{Wage} \rangle \langle \text{Stimulus} \rangle}{n - \sum_{i=1}^{n} p_i}, \text{if } \ p_i = 0 \end{cases},$$

where $P = [p_1, \cdots, p_n]$, with $p_i \in \{0, 1\}$ is the cue pattern for the pattern that is to be retrieved.
All quantities enclosed in angled brackets are system parameters, and LTI updating is accomplished using a completely analogous set of equations.

The changing STI values then cause updating of the connection matrix, according to the "conjunction" equations. First define

$$\text{norm}_i = \begin{cases} \frac{s_i}{\text{recentMaxSTI}}, \text{if } s_i \geq 0 \\ \frac{s_i}{\text{recentMinSTI}}, \text{if } s_i < 0 \end{cases}.$$

Next define

$$\text{conj} = \text{Conjunction}\,(\text{s}_\text{i}, s_j) = \text{norm}_\text{i} \times \text{norm}_\text{j}$$

and

$$c'_{ij} = \langle \text{ConjDecay} \rangle \, \text{conj} + (1 - \text{conj})\, c_{ij}.$$

Finally update the matrix elements by setting

$$c_{ij} = \begin{cases} c_{ji} = c'_{ij}, \text{if } \ c'_{ij} \geq 0 \\ c'_{ij}, \qquad \text{if } \ c'_{ij} < 0 \end{cases}.$$

We are currently also experimenting with updating the connection matrix in accordance with the equations suggested by Storkey (1997, 1998, 1999.)

A key property of these equations is that both wages paid to, and rent paid by, each node are positively correlated to their STI values. That is, the more

important nodes are paid more for their services, but they also pay more in rent.

A fixed percentage of the links with the lowest LTI values is then forgotten (which corresponds equationally to setting the LTI to 0).

Separately from the above, the process of Hebbian probability updating is carried out via a diffusion process in which some nodes "trade" STI utilizing a diffusion matrix D, a version of the connection matrix C normalized so that D is a left stochastic matrix. D acts on a similarly scaled vector v, normalized so that v is equivalent to a probability vector of STI values.

The decision about which nodes diffuse in each diffusion cycle is carried out via a decision function. We currently are working with two types of decision functions: a standard threshold function, by which nodes diffuse if and only if the nodes are in the AF; and a stochastic decision function in which nodes diffuse with probability $\frac{\tanh(\text{shape}(s_i - \text{FocusBoundary}))+1}{2}$, where shape and FocusBoundary are parameters.

The details of the diffusion process are as follows. First, construct the diffusion matrix from the entries in the connection matrix as follows:

$$\begin{array}{l}\text{If } c_{ij} \geq 0, \text{ then } d_{ij} = c_{ij}, \\ \text{else, set } d_{ji} = -c_{ij}.\end{array}$$

Next, we normalize the columns of D to make D a left stochastic matrix. In so doing, we ensure that each node spreads no more that a $\langle \text{MaxSpread} \rangle$ proportion of its STI, by setting

$$\text{if } \sum_{\text{i=1}}^{\text{n}} d_{ij} > \langle \text{MaxSpread} \rangle :$$

$$d_{ij} = \begin{cases} d_{ij} \times \frac{\langle \text{MaxSpread} \rangle}{\sum_{i=1}^{n} d_{ij}}, & \text{for } i \neq j \\ d_{jj} = 1 - \langle \text{MaxSpread} \rangle \end{cases}$$

else:

$$d_{jj} = 1 - \sum_{\substack{i=1 \\ i \neq j}}^{n} d_{ij}$$

Now we obtain a scaled STI vector v by setting

$$\text{minSTI} = \min_{i \in \{1,2,\cdots,n\}} s_i \text{ and maxSTI} = \max_{i \in \{1,2,\cdots,n\}} s_i$$

$$v_i = \frac{s_i - \min \text{STI}}{\max \text{STI} - \min \text{STI}}$$

The diffusion matrix is then used to update the node STIs

$$v' = Dv$$

and the STI values are rescaled to the interval [minSTI,maxSTI].

In both the rent and wage stage and in the diffusion stage, the total STI and LTI funds of the system each separately form a conserved quantity: in the case of diffusion, the vector v is simply the total STI times a probability vector. To maintain overall system funds within homeostatic bounds, a mid-cycle tax and rent-adjustment can be triggered if necessary; the equations currently used for this are

1. $\langle \text{Rent} \rangle = \frac{\text{recent stimulus awarded before update} \times \langle \text{Wage} \rangle}{\text{recent size of AF}}$;
2. $\text{tax} = \frac{x}{n}$, where x is the distance from the current AtomSpace bounds to the center of the homeostatic range for AtomSpace funds;
3. $\forall i: s_i = s_i - \text{tax}$

23.6.1.2 Investigation of Convergence Properties of Variant 1

Now we investigate some of the properties that the above ECAN equations display when we use an ECAN defined by them as an associative memory network in the manner of a Hopfield network.

We consider a situation where the ECAN is supplied with memories via a "training" phase in which one imprints it with a series of binary patterns of the form $P = [p_1, \cdots, p_n]$, with $p_i \in \{0, 1\}$. Noisy versions of these patterns are then used as cue patterns during the retrieval process.

We obviously desire that the ECAN retrieve the stored pattern corresponding to a given cue pattern. In order to achieve this goal, the ECAN must converge to the correct fixed point.

Theorem 23.1. *For a given value of e in the STI rent calculation, there is a subset of hyperbolic decision functions for which the ECAN dynamics converge to an attracting fixed point.*

Proof. Rent is zero whenever $s_i \leq \frac{\text{recentMaxSTI}}{20}$, so we consider this case first. The updating process for the rent and wage stage can then be written as $f(s) = s + \text{constant}$. The next stage is governed by the hyperbolic decision function

$$g(s) = \frac{\tanh(\text{shape}(\text{s} - \text{FocusBoundary})) + 1}{2}.$$

The entire updating sequence is obtained by the composition $(g \circ f)(s)$, whose derivative is then

$$(g \circ f)' = \frac{\text{sech}^2(\text{f}(\text{s})) \cdot \text{shape}}{2} \cdot (1),$$

which has magnitude less than 1 whenever -2 < shape < 2. We next consider the case $s_i > \frac{\text{recentMaxSTI}}{20}$. The function f now takes the form

$$f(s) = s - \frac{\log(20s/\text{recentMaxSTI})}{2} + \text{constant},$$

and we have

$$(g \circ f)' = \frac{\text{sech}^2(\text{f}(\text{s})) \cdot \text{shape}}{2} \cdot \left(1 - \frac{1}{s}\right).$$

which has magnitude less than 1 whenever $|\text{shape}| < \left|\frac{2 \cdot \text{recentMaxSTI}}{\text{recentMaxSTI}-20}\right|$. Choosing the shape parameter to satisfy $0 < \text{shape} < \min\left(2, \left|\frac{2 \cdot \text{recentMaxSTI}}{\text{recentMaxSTI}-20}\right|\right)$ then guarantees that $\left|(g \circ f)'\right| < 1$. Finally, g ∘ f maps the closed interval $[recentMinSti, recentMaxSTI]$ into itself, so applying the Contraction Mapping Theorem completes the proof.

23.6.1.3 Definition and Analysis of Variant 2

The ECAN variant described above has performed completely acceptably in our experiments so far; however we have also experimented with an alternate variant, with different convergence properties. In Variant 2, the dynamics of the ECAN are specifically designed so that a certain conceptually intuitive function serves as a Liapunov function of the dynamics.

At a given time t, for a given Atom indexed i, we define two quantities: $OUT_i(t)$ = the total amount that Atom i pays in rent and tax and diffusion during the time-t iteration of ECAN ; $IN_i(t)$ = the total amount that Atom i receives in diffusion, stimulus and welfare during the time-t iteration of ECAN. Note that welfare is a new concept to be introduced below. We then define $DIFF_i(t) = IN_i(t) - OUT_i(t)$; and define $AVDIFF(t)$ as the average of $DIFF_i(t)$ over all i in the ECAN.

The design goal of Variant 2 of the ECAN equations is to ensure that, if the parameters are tweaked appropriately, AVDIFF can serve as a (deterministic or stochastic, depending on the details) Lyapunov function for ECAN dynamics. This implies that with appropriate parameters the ECAN dynamics will converge toward a state where AVDIFF=0, meaning that no Atom is making any profit or incurring any loss. It must be noted that this kind of convergence is not always desirable, and sometimes one might want the parameters set otherwise. But if one wants the STI components of an ECAN to converge to some specific values, as for instance in a classic associative memory application, Variant 2 can guarantee this easily.

In Variant 2, each ECAN cycle begins with rent collection and welfare distribution, which occurs via collecting rent via the Variant 1 equation, and then performing the following two steps:

- **Step A**: calculate X, defined as the positive part of the total amount by which AVDIFF has been increased via the overall rent collection process.
- **Step B**: redistribute X to needy Atoms as follows: *For each Atom z, calculate the positive part of $OUT - IN$, defined as deficit(z). Distribute $X + e$ wealth among all Atoms z, giving each Atom a percentage of X that is proportional to deficit(z), but never so much as to cause $OUT<IN$ for any Atom (the welfare being given counts toward IN).* Here $e > 0$ ensures AVDIFF decrease; $e = 0$ may be appropriate if convergence is not required in a certain situation.

Step B is the welfare step, which guarantees that rent collection will decrease AVDIFF. Step A calculates the amount by which the rich have been made poorer, and uses this to make the poor richer. In the case that the sum of *deficit*(z) over all nodes z is less than X, a mid-cycle rent adjustment may be triggered, calculated so that step B will decrease AVDIFF. (I.e. we cut rent on the rich, if the poor don't need their money to stay out of deficit.)

Similarly, in each Variant 2 ECAN cycle, there is a wage-paying process, which involves the wage-paying equation from Variant 1 followed by two steps. Step A: calculate Y, defined as the positive part of the total amount by which AVDIFF has been increased via the overall wage payment process. Step B: exert taxation based on the surplus Y as follows: *For each Atom z, calculate the positive part of $IN - OUT$, defined as surplus(z). Collect $Y + e1$ wealth from all Atom z, collecting from each node a percentage of Y that is proportional to surplus(z), but never so much as to cause $IN < OUT$ for any node (the new STI being collected counts toward OUT).*

In case the total of *surplus*(z) over all nodes z is less than Y, one may trigger a mid-cycle wage adjustment, calculated so that step B will decrease AVDIFF. I.e. we cut wages since there is not enough surplus to support it.

Finally, in the Variant 2 ECAN cycle, diffusion is done a little differently, via iterating the following process: If AVDIFF has increased during the diffusion round so far, then choose a random node whose diffusion would decrease AVDIFF, and let it diffuse; if AVDIFF has decreased during the diffusion round so far, then choose a random node whose diffusion would increase AVDIFF, and let it diffuse. In carrying out these steps, we avoid letting the same node diffuse twice in the same round. This algorithm does not let all Atoms diffuse in each cycle, but it stochastically lets a lot of diffusion happen in a way that maintains AVDIFF constant. The iteration may be modified to bias toward an average decrease in AVDIFF.

The random element in the diffusion step, together with the logic of the rent/welfare and wage/tax steps, combines to yield the result that for Variant 2 of ECAN dynamics, AVDIFF is a stochastic Lyaponov function. The details of the proof of this will be omitted but the outline of the argument should be clear from the construction of Variant 2. And note that by setting the e and e_1 parameter to 0, the convergence requirement can be eliminated, allowing the network to evolve more spontaneously as may be appropriate in some

contexts; these parameters allow one to explicitly adjust the convergence rate.

One may also derive results pertaining to the meaningfulness of the attractors, in various special cases. For instance, if we have a memory consisting of a set M of m nodes, and we imprint the memory on the ECAN by stimulating m nodes during an interval of time, then we want to be able to show that the condition where precisely those m nodes are in the AF is a fixed-point attractor. However, this is not difficult, because one must only show that if these m nodes and none others are in the AF, this condition will persist.

23.6.2 ECAN as Associative Memory

We have carried out experiments gauging the performance of Variant 1 of ECAN as an associative memory, using the implementation of ECAN within CogPrime , and using both the conventional and Storkey Hebbian updating formulas.

As with a Hopfield net memory, the memory capacity (defined as the number of memories that can be retrieved from the network with high accuracy) depends on the sparsity of the network, with denser networks leading to greater capacity. In the ECAN case the capacity also depends on a variety of parameters of the ECAN equations, and the precise unraveling of these dependencies is a subject of current research. However, one interesting dependency has already been uncovered in our preliminary experimentation, which has to do with the size of the AF versus the size of the memories being stored.

Define the size of a memory (a pattern being imprinted) as the number of nodes that are stimulated during imprinting of that memory. In a classical Hopfield net experiment, the mean size of a memory is usually around, say, .2-.5 of the number of neurons. In typical CogPrime associative memory situations, we believe the mean size of a memory will be one or two orders of magnitude smaller than that, so that each memory occupies only a relatively small portion of the overall network.

What we have found is that the memory capacity of an ECAN is generally comparable to that of a Hopfield net with the same number of nodes and links, if and only if the ECAN parameters are tuned so that the memories being imprinted can fit into the AF. That is, the AF threshold or (in the hyperbolic case) shape parameter must be tuned so that the size of the memories is not so large that the active nodes in a memory cannot stably fit into the AF. This tuning may be done adaptively by testing the impact of different threshold/shape values on various memories of the appropriate size; or potentially a theoretical relationship between these quantities could be derived, but this has not been done yet. This is a reasonably satisfying result given the cognitive foundation of ECAN: in loose terms what it means

is that ECAN works best for remembering things that fit into its focus of attention during the imprinting process.

23.7 Glocal Economic Attention Networks

In order to transform ordinary ECANs into glocal ECANs, one may proceed in essentially the same manner as with glocal Hopfield nets as discussed in Chapter 13 of Part 1. In the language normally used to describe CogPrime , this would be termed a "map encapsulation" heuristic. As with glocal Hopfield nets, one may proceed most simply via creating a fixed pool of nodes intended to provide locally-representative keys for the maps formed as attractors of the network. Links may then be formed to these key nodes, with weights and STI and LTI values adapted by the usual ECAN algorithms.

23.7.1 Experimental Explorations

To compare the performance of glocal ECANs with glocal Hopfield networks in a simple context, we ran experiments using ECAN in the manner of a Hopfield network. That is, a number of nodes take on the equivalent role of the neurons that are presented patterns to be stored. These patterns are imprinted by setting the corresponding nodes of active bits to have their STI within the AF, whereas nodes corresponding to inactive bits of the pattern are below the AF threshold. Link weight updating then occurs, using one of several update rules, but in this case the update rule of [SV99] was used. Attention spread used a diffusion approach by representing the weights of Hebbian links between pattern nodes within a left stochastic Markov matrix, and multiplying it by the vector of normalised STI values to give a vector representing the new distribution of STI.

To explore the effects of key nodes on ECAN Hopfield networks, in [Goe08b] we used the palimpsest testing scenario of [SV99], where all the local neighbours of the imprinted pattern, within a single bit change, are tested. Each neighbouring pattern is used as input to try and retrieve the original pattern. If all the retrieved pattern are the same as the original (within a given tolerance) then the pattern is deemed successfully retrieved and recall of the previous pattern is attempted via its neighbours. The number of patterns this can repeat for successfully is called the palimpsest storage of the network.

As an example, consider one simple experiment that was run with recollection of 10×10 pixel patterns (so, 100 nodes, each corresponding to a pixel in the grid), a Hebbian link density of 30%, and with 1% of links being forgotten before each pattern is imprinted. The results demonstrated that, when the

mean palimpsest storage is calculated for each of 0, 1, 5 and 10 key nodes we find that the storage is 22.6, 22.4, 24.9, and 26.0 patterns respectively, indicating that key nodes do improve memory recall on average.

23.8 Long-Term Importance and Forgetting

Now we turn to the forgetting process (carried out by the Forgetting MindAgent), which is driven by LTI dynamics, but has its own properties as well.

Overall, the goal of the "forgetting" process is to maximize the total utility of the Atoms in the AtomSpace throughout the future. The most basic heuristic toward this end is to remove the Atoms with the lowest LTI, but this isn't the whole story. Clearly, the decision to remove an Atom from RAM should depend on factors beyond just the LTI of the Atom. For example, one should also take into account the expected difficulty in reconstituting the given Atom from other Atoms. Suppose the system has the relations:

```
''dogs are animals''

''animals are cute''

''dogs are cute''
```

and the strength of the third relation is not dissimilar from what would be obtained by deduction and revision from the first two relations and others in the system. Then, even if the system judges it will be very useful to know *dogs are cute* in the future, it may reasonably choose to remove *dogs are cute* from memory anyway, because it knows it can be so easily reconstituted, by a few inference steps for instance. Thus, as well as removing the lowest-LTI Atoms, the Forgetting MindAgent should also remove Atoms meeting certain other criteria such as the combination of:

- low STI
- easy reconstitutability in terms of other Atoms that have LTI not less than its own

23.9 Attention Allocation via Data Mining on the System Activity Table

In this section we'll discuss an object called the System Activity Table, which contains a number of subtables recording various activities carried out by the various objects in the CogPrime system. These tables may be used for sophisticated attention allocation processes, according to an approach in which importance values and HebbianLink weight values are calculated via direct

data mining of a centralized knowledge store (the System Activity Table). This approach provides highly accurate attention allocation but at the cost of significant computational effort.

The System Activity Table is actually a set of tables, with multiple components. The precise definition of the tables will surely be adapted based on experience as the work with CogPrime progresses; what is described here is a reasonable first approximation.

First, there is a MindAgent Activity Table, which includes, for each MindAgent in the system, a table such as Table 23.1 (in which the time-points recorded are the last T system cycles, and the Atom-lists recorded are lists of Handles for Atoms).

System Cycle	Effort Spent	Memory Used	Atom Combo 1 Utilized	Atom Combo 2 Utilized	...
Now	3.3	4000	Atom21, Atom44	Atom 44, Atom 47, Atom 345	...
Now -1	0.4	6079	Atom123, Atom33	Atom 345	...
...	...	...	...	...	...

Table 23.1 Example MindAgent Table

The MindAgent's activity table records, for that MindAgent and for each system cycle, which Atom-sets were acted on by that MindAgent at that point in time.

Similarly, a table of this nature must be maintained for each Task-type, e.g. InferenceTask, MOSESCategorizationTask, etc. The Task tables are used to estimate Effort values for various Tasks, which are used in the procedure execution process. If it can be estimated how much spatial and temporal resources a Task is likely to use, via comparison to a record of previous similar tasks (in the Task table), then a MindAgent can decide whether it is appropriate to carry out this Task (versus some other one, or versus some simpler process not requiring a Task) at a given point in time, a process to be discussed in a later chapter.

In addition to the MindAgent and Task-type tables, it is convenient if tables are maintained corresponding to various goals in the system (as shown in Table 23.9), including the Ubergoals but also potentially derived goals of high importance.

System Cycle	Total Achievement	Achievement for Atom44	Achievement for set {Atom44, Atom 233}	...
Now	.8	.4	.5	...
Now-1	.9	.5	.55	...
...	...	...	...	...

Table 23.2 Example Goal Table

For each goal, at minimum, the degree of achievement of the goal at a given time must be recorded. Optionally, at each point in time, the degree of achievement of a goal relative to some particular Atoms may be recorded. Typically the list of Atom-specific goal-achievements will be short and will be different for different goals and different time points. Some goals may be applied to specific Atoms or Atom sets, others may only be applied more generally.

The basic idea is that attention allocation and credit assignment may be effectively carried out via datamining on these tables.

23.10 Schema Credit Assignment

And, how do we apply a similar approach to clarifying the semantics of schema credit assignment?

From the above-described System Activity Tables, one can derive information of the form

```
Achieve(G,E,T) = ``Goal G was achieved to extent E at time T''
```

which may be grounded as, for example:

```
Similarity
    E
    ExOut
        GetTruthValue
        Evaluation
            atTime
                T
                HypLink G
```

and more refined versions such as

```
Achieve(G,E,T,A,P) = ``Goal G was achieved to extent E using
                     Atoms A (with parameters P) at time T''

Enact(S,I,$T_1$,O,$T_2$) = ``Schema S was enacted on inputs I
                           at time $T_1$, producing outputs O
                           at time $T_2$''
```

The problem of schema credit assignment is then, in essence: Given a goal G and a distribution of times $\mathcal{D}$, figure out what schema to enact in order to cause G's achievement at some time in the future, where the desirability of times is weighted by $\mathcal{D}$.

The basic method used is the learning of predicates of the form

`ImplicationLink`
    $F(C, P_1, ..., P_n)$
    $\mathcal{G}$

where

- the P_i are *Enact*() statements in which the T_1 and T_2 are variable, and the S, I and O may be concrete or variable
- C is a predicate representing a *context*
- $\mathcal{G}$ is an *Achieve*() statement, whose arguments may be concrete or abstract
- F is a Boolean function

Typically, the variable expressions in the T_1 and T_2 positions will be of the form $T + \mathit{offset}$, where offset is a constant value and T is a time value representing the time of inception of the whole compound schema. T may then be defined as $T_G - \mathit{offset}_1$, where offset_1 is a constant value and T_G is a variable denoting the time of achievement of the goal.

In CogPrime , these predicates may be learned by a combination of statistical pattern mining, PLN inference and MOSES or hill-climbing procedure learning.

The choice of what action to take at a given point in time is then a probabilistic decision. Based on the time-distribution $\mathcal{D}$ given, the system will know a certain number of expressions $\mathcal{C} = F(C, P_1, ..., P_n)$ of the type described above. Each of these will be involved in an ImplicationLink with a certain estimated strength. It may select the "compound schema" $\mathcal{C}$ with the highest strength.

One might think to introduce other criteria here, e.g. to choose the schema with the highest strength but the lowest cost of execution. However, it seems better to include all pertinent criteria in the goal, so that if one wants to consider cost of execution, one assumes the existence of a goal that incorporates cost of execution (which may be measured in multiple ways, of course) as part of its internal evaluation function.

Another issue that arises is whether to execute multiple $\mathcal{C}$ simultaneously. In many cases this won't be possible because two different $\mathcal{C}$'s will contradict each other. It seems simplest to assume that $\mathcal{C}$'s that can be fused together into a single plan of action, are presented to the schema execution process as a single fused $\mathcal{C}$. In other words, the fusion is done during the schema learning process rather than the execution process.

A question emerges regarding how this process deals with false causality, e.g. with a schema that, due to the existence of a common cause, often happens to occur immediately prior to the occurrence of a given goal. For instance, roosters crowing often occurs prior to the sun rising. This matter is discussed in more depth in the PLN book and *The Hidden Pattern*; but in brief, the answer is: In the current approach, if roosters crowing often causes the sun to rise, then if the system wants to cause the sun to rise, it may well cause a rooster to crow. Once this fails, then the system will no longer hold the false belief, and afterwards will choose a different course of action. Furthermore, if it holds background knowledge indicating that roosters crowing is not likely to cause the sun to rise, then this background knowledge will be invoked by inference to discount the strength of the ImplicationLink point-

ing from rooster-crowing to sun-rising, so that the link will never be strong enough to guide schema execution in the first place.

The problem of credit assignment thus becomes a problem of creating appropriate heuristics to guide inference of ImplicationLinks of the form described above. Assignment of credit is then implicit in the calculation of truth values for these links. The difficulty is that the predicates F involved may be large and complex.

23.11 Interaction between ECANs and other CogPrime Components

We have described above a number of interactions between attention allocation and other aspects of CogPrime ; in this section we gather a few comments on these interactions, and some additional ones.

23.11.1 Use of PLN and Procedure Learning to Help ECAN

MOSES or hillclimbing may be used to help mine the SystemActivityTable for patterns of usefulness, and create HebbianLinks reflecting these patterns.

PLN inference may be carried out on HebbianLinks by treating (HebbianLink A B) as a virtual predicate evaluation relationship , i.e. as

```
EvaluationLink Hebbian_predicate (A, B)
```

PLN inference on HebbianLinks may then be used to update node importance values, because node importance values are essentially *node probabilities* corresponding to HebbianLinks. And similarly, MindAgent-relative node importance values are node probabilities corresponding to MindAgent-relative HebbianLinks.

Note that conceptually, the nature of this application of PLN is different from most other uses of PLN in CogPrime . Here, the purpose of PLN is not to draw conclusions about the outside world, but rather about what the system should focus its resources on in what context. PLN, used in this context, effectively constitutes a nonlinear-dynamical iteration governing the flow of *attention* through the CogPrime system.

Finally, inference on HebbianLinks leads to the emergence of maps, via the recognition of clusters in the graph of HebbianLinks.

23.11.2 Use of ECAN to Help Other Cognitive Processes

First of all, associative-memory functionality is directly important in CogPrime Prime because it is used to drive concept creation. The CogPrime heuristic called "map formation" creates new Nodes corresponding to prominent attractors in the ECAN, a step that (according to our preliminary results) not only increases the memory capacity of the network beyond what can be achieved with a pure ECAN but also enables attractors to be explicitly manipulated by PLN inference.

Equally important to associative memory is the capability of ECANs to facilitate effective allocation of the attention of other cognitive processes to appropriate knowledge items (Atoms). For example, one key role of ECANs in CogPrime is to guide the forward and backward chaining processes of PLN (Probabilistic Logic Network) inference. At each step, the PLN inference chainer is faced with a great number of inference steps (branches) from which to choose; and a choice is made using a statistical "bandit problem" mechanism that selects each possible inference step with a probability proportional to its expected "desirability." In this context, there is considerable appeal in the heuristic of weighting inference steps using probabilities proportional to the STI values of the Atoms they contain. One thus arrives at a combined PLN/ECAN dynamic as follows:

1. An inference step is carried out, involving a choice among multiple possible inference steps, which is made using STI-based weightings (and made among Atoms that LTI weightings have deemed valuable enough to remain in RAM)
2. The Atoms involved in the inference step are rewarded with STI and LTI proportionally to the utility of the inference step (how much it increases the confidence of Atoms in the system's memory)
3. The ECAN operates, and multiple Atom's importance values are updated
4. Return to Step 1 if the inference isn't finished

An analogous interplay may occur between ECANs and MOSES.

It seems intuitively clear that the same attractor-convergence properties highlighted in the above analysis of associative-memory behavior, will also be highly valuable for the application of ECANs to attention allocation. If a collection of Atoms is often collectively useful for some cognitive process (such as PLN), then the associative-memory-type behavior of ECANs means that once a handful of the Atoms in the collection are found useful in a certain inference process, the other Atoms in the collection will get their STI significantly boosted, and will be likely to get chosen in subsequent portions of that same inference process. This is exactly the sort of dynamics one would like to see occur. Systematic experimentation with these interactions between ECAN and other CogPrime processes is one of our research priorities going forwards.

23.12 MindAgent Importance and Scheduling

So far we have discussed economic transactions between Atoms and Atoms, and between Atoms and Units. MindAgents have played an indirect role, via spreading stimulation to Atoms which causes them to get paid wages by the Unit. Now it is time to discuss the explicit role of MindAgents in economic transactions. This has to do with the integration of economic attention allocation with the Scheduler that schedules the core MindAgents involved in the basic cognitive cycle.

This integration may be done in many ways, but one simple approach is:

1. When a MindAgent utilizes an Atom, this results in sending stimulus to that Atom. (Note that we don't want to make MindAgents pay for using Atoms individually; that would penalize MA's that use more Atoms, which doesn't really make much sense.)
2. MindAgents then get currency from the Lobe (as defined in Chapter 19) periodically, and get extra currency based on usefulness for goal achievement as determined by the credit assignment process. The Scheduler then gives more processor time to MindAgents with more STI.
3. However, any MindAgent with LTI above a certain minimum threshold will get some minimum amount of processor time (i.e. get scheduled at least once each N cycles).

As a final note: In a multi-Lobe Unit, the Unit may use the different LTI values of MA's in different Lobes to control the distribution of MA's among Lobes: e.g. a very important (LTI) MA might get cloned across multiple Lobes.

23.13 Information Geometry for Attention Allocation

Appendix B outlined some very broad ideas regarding the potential utilization of information geometry and related ideas for modeling cognition. In this section, we present some more concrete and detailed experiments inspired by the same line of thinking. We model CogPrime 's Economic Attention Networks (ECAN) component using information geometric language, and then use this model to propose a novel information geometric method of updating ECAN networks (based on an extension of Amari's ANGL algorithm). Tests on small networks suggest that information-geometric methods have the potential to vastly improve ECAN's capability to shift attention from current preoccupations to desired preoccupations. However, there is a high computational cost associated with the simplest implementations of these methods, which has prevented us from carrying out large-scale experiments so far. We are exploring the possibility of circumventing these issues via using sparse matrix algorithms on GPUs.

23.13.1 Brief Review of Information Geometry

"Information geometry" is a branch of applied mathematics concerned with the application of differential geometry to spaces of probability distributions. In [GI11] we have suggested some extensions to traditional information geometry aimed at allowing it to better model general intelligence. However for the concrete technical work in the present paper, the traditional formulation of information geometry will suffice.

One of the core mathematical constructs underlying information geometry, is the Fisher Information, a statistical quantity which has a a variety of applications ranging far beyond statistical data analysis, including physics [Fri98], psychology and AI [AN00]. Put simply, FI is a formal way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ upon which the probability of X depends. FI forms the basis of the Fisher-Rao metric, which has been proved the only Riemannian metric on the space of probability distributions satisfying certain natural properties regarding invariance with respect to coordinate transformations. Typically θ in the FI is considered to be a real multidimensional vector; however, [Dab99] has presented a FI variant that imposes basically no restrictions on the form of θ. Here the multidimensional FI will suffice, but the more general version is needed if one wishes to apply FI to AGI more broadly, e.g. to declarative and procedural as well as attentional knowledge.

In the set-up underlying the definition of the ordinary finite-dimensional Fisher information, the probability function for X, which is also the likelihood function for $\theta \in R^n$, is a function $f(X;\theta)$; it is the probability mass (or probability density) of the random variable X conditional on the value of θ. The partial derivative with respect to θ_i of the log of the likelihood function is called the *score* with respect to θ_i. Under certain regularity conditions, it can be shown that the first moment of the score is 0. The second moment is the Fisher information:

$$\mathcal{I}(\theta)_i = \mathcal{I}_X(\theta)_i = E\left[\left(\left(\frac{\partial}{\partial\theta_i}\ln f(X;\theta)\right)^2\right)|\theta\right]$$

where, for any given value of θ_i, the expression $E[..|\theta]$ denotes the conditional expectation over values for X with respect to the probability function $f(X;\theta)$ given θ. Note that $0 \leq \mathcal{I}(\theta)_i < \infty$. Also note that, in the usual case where the expectation of the score is zero, the Fisher information is also the variance of the score.

One can also look at the whole Fisher information matrix

$$\mathcal{I}(\theta)_{i,j} = E\left[\left(\frac{\partial ln f(X,\theta)}{\partial\theta_i}\frac{\partial ln f(X,\theta)}{\partial\theta_j}\right)|\theta\right]$$

which may be interpreted as a metric g_{ij}, that provably is the only "intrinsic" metric on probability distribution space. In this notation we have $\mathcal{I}(\theta)_i = \mathcal{I}(\theta)_{i,i}$.

Dabak [Dab99] has shown that the geodesic between two parameter vectors θ and θ' is given by the exponential weighted curve $(\gamma(t))(x) = \frac{f(x,\theta)^{1-t} f(x,\theta')^t}{\int f(y,\theta)^{1-t} f(y,\theta')^t dy}$, under the weak condition that the log-likelihood ratios with respect to $f(X,\theta)$ and $f(X,\theta')$ are finite. Also, along this sort of curve, the sum of the Kullback-Leibler distances between θ and θ', known as the J-divergence, equals the integral of the Fisher information along the geodesic connecting θ and θ'.

This suggests that if one is attempting to learn a certain parameter vector based on data, and one has a certain other parameter vector as an initial value, it may make sense to use algorithms that try to follow the Fisher-Rao geodesic between the initial condition and the desired conclusion. This is what Amari [Ama85] [AN00] calls "natural gradient" based learning, a conceptually powerful approach which subtly accounts for dependencies between the components of θ.

23.13.2 Information-Geometric Learning for Recurrent Networks: Extending the ANGL Algorithm

Now we move on to discuss the practicalities foinformation-geometric learning within CogPrime 's ECAN component. As noted above, Amari [Ama85, AN00] introduced the natural gradient as a generalization of the direction of steepest descent on the space of loss functions of the parameter space. Issues with the original implementation include the requirement of calculating both the Fisher information matrix and its inverse. To resolve these and other practical considerations, Amari [Ama98] proposed an adaptive version of the algorithm, the Adaptive Natural Gradient Learning (ANGL) algorithm. Park, Amari, and Fukumizu [PAF00] extended ANGL to a variety of stochastic models including stochastic neural networks, multi-dimensional regression, and classification problems.

In particular, they showed that, assuming a particular form of stochastic feedforward neural network and under a specific set of assumptions concerning the form of the probability distributions involved, a version of the Fisher information matrix can be written as

$$G(\theta) = E_\xi \left[\left(\frac{r'}{r} \right)^2 \right] E_x \left[\nabla H \left(\nabla H \right)^T \right].$$

Although Park et al considered only feedforward neural networks, their result also holds for more general neural networks, including the ECAN net-

work. What is important is the decomposition of the probability distribution as

$$p\left(\mathbf{y}|\mathbf{x};\theta\right)=\prod_{i=1}^{L}r_i\left(y_i-H_i\left(\mathbf{x},\theta\right)\right)$$

where

$$\mathbf{y}=\mathbf{H}(\mathbf{x};\theta)+\xi,\ \mathbf{y}=(y_1,\cdots,y_L)^T,\ \mathbf{H}=(H_1,\cdots,H_L)^T,\ \xi=(\xi_1,\cdots,\xi_L)^T,$$

where ξ is added noise. If we assume further that each r_i has the same form as a Gaussian distribution with zero mean and standard deviation σ, then the Fisher information matrix simplifies further to

$$G(\theta)=\frac{1}{\sigma^2}E_x\left[\nabla H\left(\nabla H\right)^T\right].$$

The adaptive estimate for $\hat{G}_{t+1}^{-1}$ is given by

$$\hat{G}_{t+1}^{-1}=(1+\epsilon_t)\hat{G}_t^{-1}-\epsilon_t(\hat{G}_t^{-1}\nabla H)(\hat{G}_t^{-1}\nabla H)^T.$$

and the loss function for our model takes the form

$$l(\mathbf{x},\mathbf{y};\theta)=-\sum_{\mathbf{i=1}}^{\mathbf{L}}\log\mathbf{r}(\mathbf{y_i}-\mathbf{H_i}(\mathbf{x},\theta)).$$

The learning algorithm for our connection matrix weights θ is then given by

$$\theta_{t+1}=\theta_t-\eta_t\hat{G}_t^{-1}\nabla l(\theta_t).$$

23.13.3 Information Geometry for Economic Attention Allocation: A Detailed Example

We now present the results of a series of small-scale, exploratory experiments comparing the original ECAN process running alone with the ECAN process coupled with ANGL. We are interested in determining which of these two lines of processing result in focusing attention more accurately.

The experiment started with base patterns of various sizes to be determined by the two algorithms. In the training stage, noise was added, generating a number of instances of noisy base patterns. The learning goal is to identify the underlying base patterns from the noisy patterns as this will identify how well the different algorithms can focus attention on relevant versus irrelevant nodes.

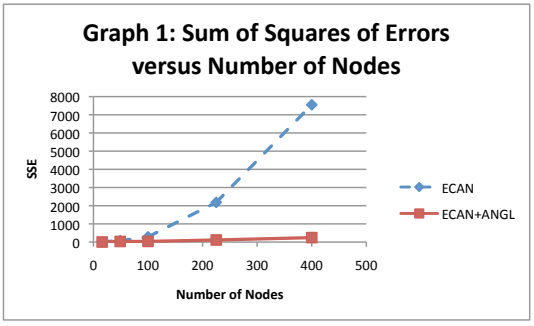


Fig. 23.3 Results from Experiment 1

Next, the ECAN process was run, resulting in the determination of the connection matrix C. In order to apply the ANGL algorithm, we need the gradient, ∇H, of the ECAN training process, with respect to the input $\mathbf{x}$. While calculating the connection matrix C, we used Monte Carlo simulation to simultaneously calculate an approximation to ∇H.

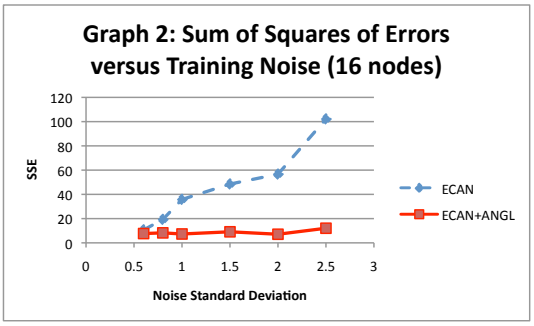


Fig. 23.4 Results from Experiment 2

After ECAN training was completed, we bifurcated the experiment. In one branch, we ran fuzzed cue patterns through the retrieval process. In the other, we first applied the ANGL algorithm, optimizing the weights in the connection matrix, prior to running the retrieval process on the same fuzzed cue patterns. At a constant value of $\sigma = 0.8$ we ran several samples through each branch with pattern sizes of 4×4, 7×7, 10×10, 15×15, and 20×20. The results are shown in Figure 23.3. We also ran several experiments comparing the sum of squares of the errors to the input training noise as measured by the value of σ.; see Figures 23.4 and **??**.

These results suggest two major advantages of the ECAN+ANGL combination compared to ECAN alone. Not only was the performance of the combination better in every trial, save for one involving a small number of

nodes and little noise, but the combination clearly scales significantly better both as the number of nodes increases, and as the training noise increases.

Chapter 24
Economic Goal and Action Selection

24.1 Introduction

A significant portion of CogPrime 's dynamics is explicitly goal-driven – that is, based on trying (inasmuch as possible within the available resources) to figure out which actions will best help the system achieve its goals, given the current context. A key aspect of this explicit activity is guided by the process of "goal and action selection" – prioritizing goals, and then prioritizing actions based on these goals. We have already outlined the high-level process of action selection, in Chapter 22 above. Now we dig into the specifics of the process, showing how action selection is dynamically entwined with goal prioritization, and how both processes are guided by economic attention allocation as described in Chapter 23.

While the basic structure of CogPrime 's action selection aspect is fairly similar to MicroPsi (due to the common foundation in Dorner's Psi model), the dynamics are less similar. MicroPsi's dynamics are a little closer to being a formal neural net model, whereas ECAN's economic foundation tends to push it in different directions. The CogPrime goal and action selection design involves some simple simulated financial mechanisms, building on the economic metaphor of ECAN, that are different from, and more complex than, anything in MicroPsi.

The main actors (apart from the usual ones like the AtomTable, economic attention allocation, etc.) in the tale to be told here are as follows:

- Structures:
  - UbergoalPool
  - ActiveSchemaPool
- MindAgents:
  - GoalBasedSchemaSelection
  - GoalBasedSchemaLearning

- GoalAttentionAllocation
- FeasibilityUpdating
- SchemaActivation

The Ubergoal Pool contains the Atoms that the system considers as top-level goals. These goals must be treated specially by attention allocation: they must be given funding by the Unit so that they can use it to pay for getting themselves achieved. The weighting among different top-level goals is achieved via giving them differential amounts of currency. STICurrency is the key kind here, but of course ubergoals must also get some LTICurrency so they won't be forgotten. (Inadvertently deleting your top-level supergoals from memory is generally considered to be a bad thing ... it's in a sense a sort of suicide...)

24.2 Transfer of STI "Requests for Service" Between Goals

Transfer of "attentional funds" from goals to subgoals, and schema modules to other schema modules in the same schema, take place via a mechanism of promises of funding (or 'requests for service,' to be called 'RFS's' from here on). This mechanism relies upon and interacts with ordinary economic attention allocation but also has special properties. Note that we will sometimes say that an Atom "issues" RFS or "transfers" currency while what we really mean is that some MindAgent working on that Atom issues RFS or transfers currency.

The logic of these RFS's is as follows. If agent A issues a RFS of value x to agent B, then

1. When B judges it appropriate, B may redeem the note and ask A to transfer currency of value x to B.
2. A may withdraw the note from B at any time.

(There is also a little more complexity here, in that we will shortly introduce the notion of RFS's whose value is defined by a set of constraints. But this complexity does not contradict the two above points.) The total value of the of RFS's possessed by an Atom may be referred to as its 'promise.'

A rough schematic depiction of this RFS process is given in Figure 24.1.

Now we explain how RFS's may be passed between goals. Given two predicates A and B, if A is being considered as a goal, then B may be considered as a subgoal of A (and A the supergoal of B) if there exists a Link of the form

```
PredictiveImplication B A
```

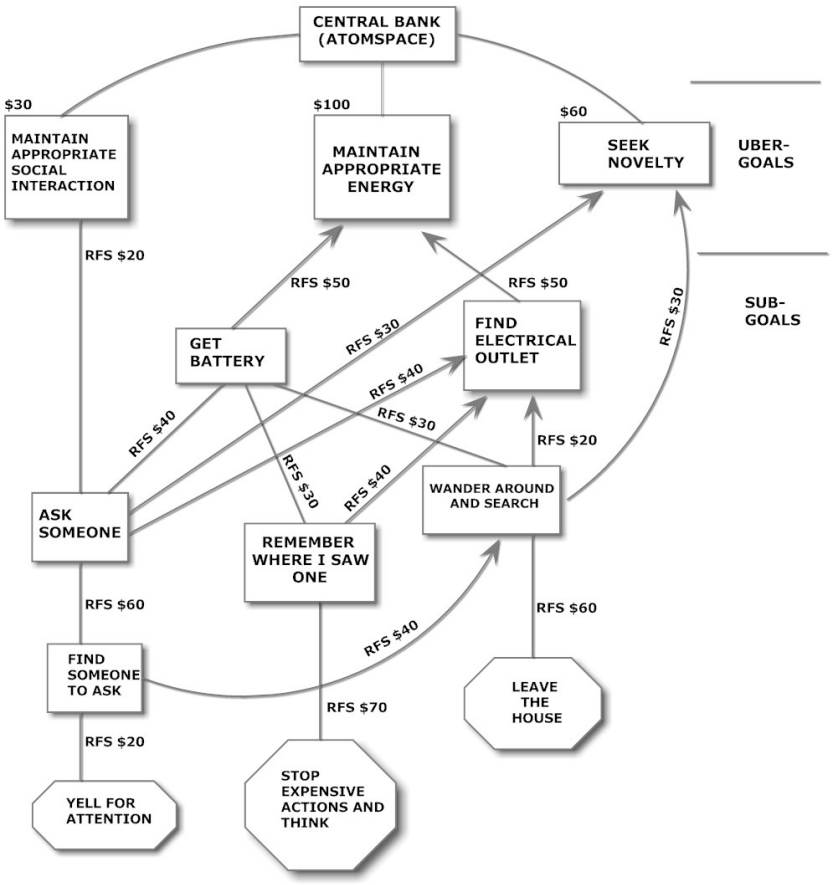


Fig. 24.1 The RFS Propagation Process. An illustration of the process via which RFS's propagate from goals to abstract procedures, and finally must get cashed out to pay for the execution of actual concrete procedures that are estimated relatively likely to lead to goal fulfillment.

I.e., achieving B may help to achieve A. Of course, the strength of this link and the temporal characteristics of this link are important in terms of quantifying how strongly and how usefully B is a subgoal of A.

Supergoals (not only top-level ones, aka ubergoals) allocate RFS's to subgoals as follows. Supergoal A may issue a RFS to subgoal B if it is judged that achievement (i.e., predicate satisfaction) of B implies achievement of A. This may proceed recursively: subgoals may allocate RFS's to subsubgoals according to the same justification.

Unlike actual currency, RFS's are not conserved. However, the actual payment of real currency upon redemption of RFS's obeys the conservation of real currency. This means that agents need to be responsible in issuing and

withdrawing RFS's. In practice this may be ensured by having agents follow a couple simple rules in this regard.

1. If B and C are two alternatives for achieving A, and A has x units of currency, then A may promise both B and C x units of currency. Whomever asks for a redemption of the promise first, will get the money, and then the promise will be rescinded from the other one.
2. On the other hand, if the achievement of A requires both B and C to be achieved, then B and C may be granted RFS's that are defined by constraints. If A has x units of currency, then B and C receive an RFS tagged with the constraint (B+C<x). This means that in order to redeem the note, either one of B or C must confer with the other one, so that they can simultaneously request constraint-consistent amounts of money from A.

As an example of the role of constraints, consider the goal of playing fetch successfully (a subgoal of "get reward").... Then suppose it is learned that:

```
ImplicationLink
    SequentialAND
         get_ball
         deliver_ball
    play_fetch
```

where *SequentialAND A B* is the conjunction of A and B but with B occurring after A in time. Then, if play_fetch has \$10 in STICurrency, it may know it has \$10 to spend on a combination of get_ball and deliver_ball. In this case both get_ball and deliver_ball would be given RFS's labeled with the contraint:

```
RFS.get_ball + RFS.deliver_ball <= 10
```

The issuance of RFS's embodying constraints is different from (and generally carried out prior to) the evaluation of whether the constraints can be fulfilled.

An ubergoal may rescind offers of reward for service at any time. And, generally, if a subgoal gets achieved and has not spent all the money it needed, the supergoal will not offer any more funding to the subgoal (until/unless it needs that subgoal achieved again).

As there are no ultimate sources of RFS in OCP besides ubergoals, promise may be considered as a measure of "goal-related importance."

Transfer of RFS's among Atoms is carried out by the GoalAttentionAllocation MindAgent.

24.3 Feasibility Structures

Next, there is a numerical data structure associated with goal Atoms, which is called the feasibility structure. The feasibility structure of an Atom G

indicates the feasibility of achieving G as a goal using various amounts of effort. It contains triples of the form (t,p,E) indicating the truth value t of achieving goal G to degree p using effort E. Feasibility structures must be updated periodically, via scanning the links coming into an Atom G; this may be done by a FeasibilityUpdating MindAgent. Feasibility may be calculated for any Atom G for which there are links of the form:

```
Implication
    Execution S
    G
```

for some S. Once a schema has actually been executed on various inputs, its cost of execution on other inputs may be empirically estimated. But this is not the only case in which feasibility may be estimated. For example, if goal G inherits from goal G1, and most children (e.g. subgoals) of G1 are achievable with a certain feasibility, then probably G is achievable with a similar feasibility as well. This allows feasibility estimation even in cases where no plan for achieving G yet exists, e.g. if the plan can be produced via predicate schematization, but such schematization has not yet been carried out.

Feasibility then connects with importance as follows. Important goals will get more STICurrency to spend, thus will be able to spawn more costly schemata. So, the GoalBasedSchemaSelection MindAgent, when choosing which schemata to push into the ActiveSchemaPool, will be able to choose more costly schemata corresponding to goals with more STICurrency to spend.

24.4 Goal Based Schema Selection

Next, the GoalBasedSchemaSelection (GBSS) selects schemata to be placed into the ActiveSchemaPool. It does this by choosing goals G, and then choosing schemata that are alleged to be useful for achieving these goals. It chooses goals via a fitness function that combines promise and feasibility. This involves solving an optimization problem: figuring out how to maximize the odds of getting a lot of goal-important stuff done within the available amount of (memory and space) effort. Potentially this optimization problem can get quite subtle, but initially some simple heuristics are satisfactory. (One subtlety involves handling dependencies between goals, as represented by constraint-bearing RFS's.)

Given a goal, the GBSS MindAgent chooses a schema to achieve that goal via the heuristic of selecting the one that maximizes a fitness function balancing the estimated effort required to achieve the goal via executing the schema, with the estimated probability that executing the schema will cause the goal to be achieved.

When searching for schemata to achieve G, and estimating their effort, one factor to be taken into account is the set of schemata already in the ActiveSchemaPool. Some schemata S may simultaneously achieve two goals; or two schemata achieving different goals may have significant overlap of modules. In this case G may be able to get achieved using very little or no effort (no additional effort, if there is already a schema S in the ActiveSchemaPool that is going to cause G to be achieved). But if G "decides" it can be achieved via a schema S already in the ActiveSchemaPool, then it should still notify the ActiveSchemaPool of this, so that G can be added to S's index (see below). If the other goal G1 that placed S in the ActiveSchemaPool decides to withdraw S, then S may need to hit up G1 for money, in order to keep itself in the ActiveSchemaPool with enough funds to actually execute.

24.4.1 A Game-Theoretic Approach to Action Selection

Min Jiang has observed that, mathematically, the problem of action selection (represented in CogPrime as the problem of goal-based schema selection) can be modeled in terms of game theory, as follows:

- the intelligent agent is one player, the world is another player
- the agent's model of the world lets it make probabilistic predictions of how the world may respond to what the agent does (i.e. to estimate what mixed strategy the world is following, considering the world as a game player)
- the agent itself chooses schema probabilistically, so it's also following a mixed strategy
- so, in principle the agent can choose schema that it thinks will lead to a mixed Nash equilibrium[1]

But the world's responses are very high-dimensional, which means that finding a mixed Nash equilibrium even approximately is a very hard computational problem. Thus, in a sense, the crux of the problem seems to come down to **feature identification**. If the world's response (real or predicted) can be represented as a low-dimensional set of features, then these features can be considered as the world's "move" in the game ... and the game theory problem becomes tractable via approximation schemes. But without the reduction of the world to a low-dimensional set of features, finding the mixed Nash equilbrium even approximately will not be computationally tractable...

Some AI theorists would argue that this division into "feature identification" versus "action selection" is unnecessarily artificial; for instance, Hawkins [?] or Arel [?] might suggest to use a single hierarchical neural network to do

[1] in game theory, a Nash equilibrium is when no player can do better by unilaterally changing its strategy

both of them. But the brain after all contains many different regions, with different architectures and dynamics.... In the visual cortex, it seems that feature extraction and object classification are done separately. And it seems that in the brain, action selection has a lot to do with the basal ganglia, whereas feature extraction is done in the cortex. So the neural analogy provides some inspiration for an architecture in which feature identification and action selection are separated.

There is a literature discussing numerical methods for calculating approximate Nash equilibria, e.g. [**?**] [http://www.springerlink.com/content/p08jqj7w7rklb3r5/]; however, this is an extremely tricky topic in the CogPrime context because action selection must generally be done in real-time. Like perception processing, this may be an area calling for the use of parallel processing hardware. For instance, a neural network algorithm for finding mixed Nash equilibria could be implemented on a GPU supercomputer, enabling rapid real-time action selection based on a reduced-dimensionality model of the world produced by intelligent feature identification.

Consideration of the application of game theory in this context brings out an important point, which is that to do reasonably efficient and intelligent action selection, the agent needs some rapidly-evaluable model of the world, i.e. some way to rapidly evaluate **the predicted response of the world to a hypothetical action by the agent**. In the game theory approach (or any other sufficiently intelligent approach), for the agent to evaluate fitness of a schema-set S for achieving certain goals in a certain context, it has to (explicitly or implicitly) estimate

- how the world will respond if the agent does S
- how the agent could usefully respond to the world's response (call this action-set S_1)
- how the world will respond to the agent doing S_1
- etc.

and so to rapidly evaluate the fitness of S, the agent needs to be able to quickly estimate how the world will respond. This may be done via simulation, or it may be done via inference (which however will rarely be fast enough, unless with a very accurate inference control mechanism), or it may be done by learning some compacted model of the world as represented for instance in a hierarchical neural network.

24.5 SchemaActivation

And what happens with schemata that are actually in the ActiveSchemaPool? Let us assume that each of these schema is a collection of modules (subprograms), connected via ActivationLinks, which have semantics: (ActivationLink A B) means that if the schema that placed module A in the schema

pool is to be completed, then after A is activated, B should be activated. (We will have more to say about schemata, and their modularization, in the following chapter.)

When a goal places a schema in the ActiveSchemaPool, it grants that schema an RFS equal in value to the total or some fraction of the promissory+real currency it has in its possession. The heuristics for determining how much currency to grant may become sophisticated; but initially we may just have a goal give a schema all its promissory currency; or in the case of a top-level supergoal, all its actual currency.

When a module within a schema actually executes, then it must redeem some of its promissory currency to turn it into actual currency, because executing costs money (paid to the Lobe). Once a schema is done executing, if it hasn't redeemed all its promissory currency, it gives the remainder back to the goal that placed it in the ActiveSchemaPool.

When a module finishes executing, it passes promissory currency to the other modules to which it points with ActivationLinks.

The network of modules in the ActiveSchemaPool is a digraph (whose links are ActivationLinks), because some modules may be shared within different overall schemata. Each module must be indexed via which schemata contain it, and each schema must be indexed via which goal(s) want it in the ActiveSchemaPool.

24.6 GoalBasedSchemaLearning

Finally, we have the process of trying to figure out how to achieve goals, i.e. trying to learn links between ExecutionLinks and goals G. This process should be focused on goals that have a high importance but for which feasible achievement-methodologies are not yet known. Predicate schematization is one way of achieving this; another is MOSES procedure evolution.

Chapter 25
Integrative Procedure Evaluation

25.1 Introduction

Procedural knowledge must be learned, an often subtle and difficult process – but it must also be enacted. Procedure enaction is not as tricky a topic as procedure learning, but still is far from trivial, and involves the real-time interaction of procedures, during the course of execution, with other knowledge. In this brief chapter we explain how this process may be most naturally and flexibly carried out, in the context of CogPrime 's representation of procedures as programs ("Combo trees").

While this may seem somewhat of a "mechanical," implementation-level topic, it also involves some basic conceptual points, on which CogPrime as an AGI design does procedure evaluation fundamentally differently from narrow-AI systems or conventional programming language interpreters. Basically, what makes CogPrime Combo tree evaluation somewhat subtle is due to the interfacing between the Combo evaluator itself and the rest of the CogPrime system.

In the CogPrime design, Procedure objects (which contain Combo trees, and are associated with ProcedureNodes) are evaluated by ProcedureEvaluator objects. Different ProcedureEvaluator objects may evaluate the same Combo tree in different ways. Here we explain these various sorts of evaluation – how they work and what they mean.

25.2 Procedure Evaluators

In this section we will mention three different ProcedureEvaluators: the

- Simple procedure evaluation
- Effort-based procedure evaluation, which is more complex but is required for integration of inference with procedure evaluation

- Adaptive evaluation order based procedure evaluation

In the following section we will delve more thoroughly into the interactions between inference and procedure evaluation.

Another related issue is the modularization of procedures. This issue however is actually orthogonal to the distinction between the three ProcedureEvaluators mentioned above. Modularity simply requires that particular nodes within a Combo tree be marked as "module roots", so that they may be extracted from the Combo tree as a whole and treated as separate modules (called differently, sub-routines), if the ExecutionManager judges this appropriate.

25.2.1 Simple Procedure Evaluation

The SimpleComboTreeEvaluator simply does Combo tree evaluation as described earlier. When an Atom is encountered, it looks into the AtomTable to evaluate the object.

In the case that a Schema refers to an ungrounded SchemaNode (that is not defined by a ComboTree as defined in Chapter 19), and an appropriate EvaluationLink value isn't in the AtomTable, there's an evaluation failure, and the whole procedure evaluation returns the truth value $\langle .5, 0\rangle$: i.e., a zero-weight-of-evidence truth value, which is equivalent essentially to returning no value.

In the case that an Predicate refers to an ungrounded PredicateNode, and an appropriate EvaluationLink isn't in the AtomTable, then some very simple "default thinking" is done, and it is assigned the truth value of the predicate on the given arguments to be the TruthValue of the corresponding PredicateNode (which is defined as the mean truth value of the predicate across all arguments known to CogPrime)

25.2.2 Effort Based Procedure Evaluation

The next step is to introduce the notion of "effort" the amount of effort that the CogPrime system must undertake in order to carry out a procedure evaluation. The notion of effort is encapsulated in Effort objects, which may take various forms. The simplest Effort objects measure only elapsed processing time; more advanced Effort objects take into consideration other factors such as memory usage.

An effort-based Combo tree evaluator keeps a running total of the effort used in evaluating the Combo tree. This is necessary if inference is to be used to evaluate Predicates, Schema, Arguments, etc. Without some control

of effort expenditure, the system could do an arbitrarily large amount of inference to evaluate a single Atom.

The matter of evaluation effort is nontrivial because in many cases a given node of a Combo tree may be evaluated in more than one way, with a significant effort differential between the different methodologies. If a Combo tree Node refers to a predicate or schema that is very costly to evaluate, then the ProcedureEvaluator managing the evaluation of the Combo tree must decide whether to evaluate it directly (expensive) or estimate the result using inference (cheaper but less accurate). This decision depends on how much effort the ProcedureEvaluator has to play with, and what percentage of this effort it finds judicious to apply to the particular Combo tree Node in question.

In the relevant prototypes we built within OpenCog, this kind of decision was made based on some simple heuristics inside ProcedureEvaluator objects. However, it's clear that, in general, more powerful intelligence must be applied here, so that one needs to have ProcedureEvaluators that - in cases of subprocedures that are both important and highly expensive - use PLN inference to figure out how much effort to assign to a given subproblem.

The simplest useful kind of effort-based Combo tree evaluator is the EffortIntervalComboTreeEvaluator, which utilizes an Effort object that contains three numbers (yes, no, max). The yes parameter tells it how much effort should be expended to evaluate an Atom if there is a ready answer in the AtomTable. The no parameter tells it how much effort should be expended in the case that there is not a ready answer in the AtomTable. The max parameter tells it how much effort should be expended, at maximum, to evaluate all the Atoms in the Combo tree, before giving up. Zero effort, in the simplest case, may be heuristically defined as simply looking into the AtomTable – though in reality this does of course take effort, and a more sophisticated treatment would incorporate this as a factor as well.

Quantification of amounts of effort is nontrivial, but a simple heuristic guideline is to assign one unit of effort for each inference step. Thus, for instance,

- (yes, no, max) = (0,5,1000) means that if an Atom can be evaluated by AtomTable lookup, this is done, but if AtomTable lookup fails, a minimum of 5 inference steps are done to try to do the evaluation. It also says that no more than 1000 evaluations will be done in the course of evaluating the Combo tree.
- (yes, no, max) = (3,5,1000) says the same thing, but with the change that even if evaluation could be done by direct AtomTable lookup, 3 inference steps are tried anyway, to try to improve the quality of the evaluation.

25.2.3 Procedure Evaluation with Adaptive Evaluation Order

While tracking effort enables the practical use of inference within Combo tree evaluation, if one has truly complex Combo trees, then a higher degree of intelligence is necessary to guide the evaluation process appropriately. The order of evaluation of a Combo tree may be determined adaptively, based on up to three things:

- The history of evaluation of the Combo tree
- Past history of evaluation of other Combo tree's, as stored in a special AtomTable consisting only of relationships about Combo tree-evaluation-order probabilities
- New information entering into CogPrime 's primary AtomTable during the course of evaluation

ProcedureEvaluator objects may be selected at runtime by cognitive schemata, and they may also utilize schemata and MindAgents internally. The AdaptiveEvaluationOrderComboTreeEvaluator is more complex than the other ProcedureEvaluators discussed, and will involve various calls to CogPrime MindAgents, particularly those concerned with PLN inference. WIKISOURCE:ProcedureExecutionDetails

25.3 The Procedure Evaluation Process

Now we give a more thorough treatment of the procedure evaluation process, as embodied in the effort-based or adaptive-evaluation-order evaluators discussed above. The process of procedure evaluation is somewhat complex, because it encompasses three interdependent processes:

- The mechanics of procedure evaluation, which in the CogPrime design involves traversing Combo trees in an appropriate order. When a Combo tree node referring to a predicate or schema is encountered during Combo tree traversal, the process of predicate evaluation or schema execution must be invoked.
- The evaluation of the truth values of predicates - which involves a combination of inference and (in the case of grounded predicates) procedure evaluation.
- The computation of the truth values of schemata - which may involve inference as well as procedure evaluation.

We now review each of these processes.

25.3.1 Truth Value Evaluation

What happens when the procedure evaluation process encounters a Combo tree Node that represents a predicate or compound term? The same thing as when some other CogPrime process decides it wants to evaluate the truth value of a PredicateNode or CompoundTermNode: the generic process of predicate evaluation is initiated.

This process is carried out by a TruthValueEvaluator object. There are several varieties of TruthValueEvaluator, which fall into the following hierarchy:

```
TruthValueEvaluator
   DirectTruthValueEvaluator (abstract)
       SimpleDirectTruthValueEvaluator
   InferentialTruthValueEvaluator (abstract)
       SimpleInferentialTruthValueEvaluator
   MixedTruthValueEvaluator
```

A DirectTruthValueEvaluator evaluates a grounded predicate by directly executing it on one or more inputs; an InferentialTruthValueEvaluator evaluates via inference based on the previously recorded, or specifically elicited, behaviors of other related predicates or compound terms. A MixedTruthValueEvaluator contains references to a DirectTruthValueEvaluator and an InferentialTruthValueEvaluator, and contains a weight that tells it how to balance the outputs from the two.

Direct truth value evaluation has two cases. In one case, there is a given argument for the predicate; then, one simply plugs this argument in to the predicate's internal Combo tree, and passes the problem off to an appropriately selected ProcedureEvaluator. In the other case, there is no given argument, and one is looking for the truth value of the predicate *in general.* In this latter case, some estimation is required. It is not plausible to evaluate the truth value of a predicate on every possible argument, so one must sample a bunch of arguments and then record the resulting probability distribution. A greater or fewer number of samples may be taken, based on the amount of effort that's been allocated to the evaluation process. It's also possible to evaluate the truth value of a predicate in a given context (information that's recorded via embedding in a ContextLink); in this case, the random sampling is restricted to inputs that lie within the specified context.

On the other hand, the job of an InferentialTruthValueEvaluator is to use inference rather than direct evaluation to guess the truth value of a predicate (sometimes on a particular argument, sometimes in general). There are several different control strategies that may be applied here, depending on the amount of effort allocated. The simplest strategy is to rely on analogy, simply searching for similar predicates and using their truth values as guidance. (In the case where a specific argument is given, one searches for similar predicates that have been evaluated on similar arguments.) If more effort is available, then a more sophisticated strategy may be taken. Generally,

an InferentialTruthValueEvaluator may invoke a SchemaNode that embodies an inference control strategy for guiding the truth value estimation process. These SchemaNodes may then be learned like any others.

Finally, a MixedTruthValueEvaluator operates by consulting a DirectTruthValueEvaluator and/or an InferentialTruthValueEvaluator as necessary, and merging the results. Specifically, in the case of an ungrounded PredicateNode, it simply returns the output of the InferentialTruthValueEvaluator it has chosen. But in the case of a GroundedPredicateNode, it returns a weighted average of the directly evaluated and inferred values, where the weight is a parameter. In general, this weighting may be done by a SchemaNode that is selected by the MixedTruthValueEvaluator; and these schemata may be adaptively learned.

25.3.2 Schema Execution

Finally, schema execution is handled similarly to truth value evaluation, but it's a bit simpler in the details. Schemata have their outputs evaluated by SchemaExecutor objects, which in turn invoke ProcedureEvaluator objects. We have the hierarchy:

```
SchemaExecutor
   DirectSchemaExecutor (abstract)
       SimpleDirectSchemaExecutor
   InferentialSchemaExecutor (abstract)
       SimpleInferentialSchemaExecutor
    MixedSchemaExecutor
```

A DirectSchemaExecutor evaluates the output of a schema by directly executing it on some inputs; an InferentialSchemaExecutor evaluates via inference based on the previously recorded, or specifically elicited, behaviors of other related schemata. A MixedSchemaExecutor contains references to a DirectSchemaExecutor and an InferentialSchemaExecutor, and contains a weight that tells it how to balance the outputs from the two (not always obvious, depending on the output type in question).

Contexts may be used in schema execution, but they're used only indirectly, via being passed to TruthValueEvaluators used for evaluating truth values of PredicateNodes or CompoundTermNodes that occur internally in schemata being executed.

Section VIII
Perception and Action

Chapter 26
Perceptual and Motor Hierarchies

26.1 Introduction

Having discussed declarative, attentional, intentional and procedural knowledge, we are left only with sensorimotor and episodic knowledge to complete our treatment of the basic CogPrime "cognitive cycle" via which a CogPrime system can interact with an environment and seek to achieve its goals therein.

The cognitive cycle in its most basic form leaves out the most subtle and unique aspects of CogPrime , which all relate to learning in various forms. But nevertheless it is the foundation on which CogPrime is built, and within which the various learning processes dealing with the various forms of memory all interact. The CogPrime cognitive cycle is more complex in many respects than it would need to be if not for the need to support diverse forms of learning. And this learning-driven complexity is present to some extent in the contents of the present chapter as well. If learning weren't an issue, perception and actuation could more likely be treated as wholly (or near-wholly) distinct modules, operating according to algorithms and structures independent of cognition. But our suspicion is that this sort of approach is unlikely to be adequate for achieving high levels of perception and action capability under real-world conditions. Instead, we suspect, it's necessary to create perception and action processes that operate fairly effectively on their own, but are capable of cooperating with cognition to achieve yet higher levels of functionality.

And the benefit in such an approach goes both ways. Cognition helps perception and actuation deal with difficult cases, where the broad generalization that is cognition's specialty is useful for appropriately biasing perception and actuation based on subtle environmental regularities. And, the patterns involved in perception and actuation help cognition, via supplying a rich reservoir of structures and processes to use as analogies for reasoning and learning at various levels of abstraction. The prominence of visual and other sensory metaphors in abstract cognition is well known [Arn69, **?**]; and

according to Lakoff and Nunez [LN00] even pure mathematics is grounded in physical perception and action in very concrete ways.

We begin by discussing the perception and action mechanisms required to interface CogPrime with an agent operating in a virtual world. We then turn to the more complex mechanisms needed to effectively interface CogPrime with a robot possessing vaguely humanoid sensors and actuators, focusing largely on vision processing. This discussion leads up to deeper discussions in Chapters 27, 28 and 29 where we describe in detail the strategy that would be used to integrate CogPrime with the DeSTIN framework for AGI perception/action (which was described in some detail in Chapter 4 of Part 1).

In terms of the integrative cognitive architecture presented in Chapter 5, the material presented in the chapters in this section has mostly to do with the perceptual and motor hierarchies, also touching on the pattern recognition and imprinting processes that play a role in the interaction between these hierarchies and the conceptual memory. The commitment to a hierarchical architecture for perception and action is not critical for the CogPrime design as a whole – one could build a CogPrime with non-hierarchical perception and action modules, and the rest of the system would be about the same. The role of hierarchy here is a reflection of the obvious hierarchical structure of the everyday human environment, and of the human body. In a world marked by hierarchical structure, an hierarchically structured perceptual system is advantageous. To control a body marked by hierarchical structure, an hierarchically structured action system is advantageous. It would be possible to create a CogPrime system without this sort of in-built hierarchical structure, and have it gradually self-adapt in such a way as to grow its own internal hierarchical structure, based its experience in the world. However, this might be a case of pushing the "experiential learning" perspective too far. The human brain definitely has hierarchical structure built into it; it doesn't need to learn to experience the world in hierarchical terms; and there seems no good reason to complicate an AGIs early development phase by forcing it to learn the basic fact of the world's and it's body's hierarchality.

26.2 The Generic Perception Process

We have already discussed the generic action process of CogPrime , in Chapter 25 on procedure evaluation. Action sequences are generated by Combo programs, which execute primitive actions, including those corresponding to actuator control signals as well as those corresponding to, say, mathematical or cognitive operations. In some cases the actuator control signals may directly dictate movements; in other cases they may supply inputs and/or parameters to other software (such as DeSTIN, in the integrated CogBot architecture to be described below).

What about the generic perception process? We distinguish *sensation* from *perception*, in a CogPrime context, by defining

- *perception* as what occurs when some signal from the outside world registers itself in either: a CogPrime Atom, or some other sort of node (e.g. a DeSTIN node) that is capable of serving as the target of a CogPrime Link.
- *sensation* as any "preprocessing" that occurs between the impact of some signal on some sensor, and the creation of a corresponding perception

Once perceptual Atoms have been created, various perceptual MindAgents comes into play, taking perceptual schemata (schemata whose arguments are perceptual nodes or relations therebetween) and applying them to Atoms recently created (creating appropriate ExecutionLinks to store the results). The need to have special, often modality-specific perception MindAgents to do this, instead of just leaving it to the generic SchemaExecution MindAgent, has to do with computational efficiency, scheduling and parameter settings. Perception MindAgents are doing schema execution urgently, and doing it with parameter settings tuned for perceptual processing. This means that, except in unusual circumstances, newly received stimuli will be processed immediately by the appropriate perceptual schemata.

Some newly formed perceptual Atoms will have links to existing atoms, ready-made at their moment of creation. CharacterInstanceNodes and NumberInstanceNodes are examples; they are born linked to the appropriate CharacterNodes and NumberNodes. Of course, atoms representing perceived relationships, perceived groupings, etc., will not have ready-made links and will have to grow such links via various cognitive processes. Also, the ContextFormation MindAgent looks at perceptual atom creation events and creates Context Nodes accordingly; and this must be timed so that the Context Nodes are entered into the system rapidly, so that they can be used by the processes doing initial-stage link creation for new perceptual Atoms.

In a full CogPrime configuration, newly created perceptual nodes and perceptual schemata may reside in a special perception-oriented Units, so as to ensure that perceptual processes occur rapidly, not delayed by slower cognitive processes.

26.2.1 The ExperienceDB

Separate from the ordinary perception process, it may also valuable for there to be a direct route from the system's sensory sources to a special "ExperienceDB" database that records all of the system's experience. This does not involve perceptual schemata at all, nor is it left up to the sensory source; rather, it is carried out by the CogPrime server at the point where it receives input from the sensory source. This experience database is a record of what

the system has seen in the past, and may be mined by the system in the future for various purposes. The creation of new perceptual atoms may also be stored in the experience database, but this must be handled with care as it can pose a large computational expense; it will often be best to store only a subset of these.

Obviously, such an ExperienceDB is something that has no correlate in the human mind/brain. This is a case where CogPrime takes advantage of the non-brainlike properties of its digital computer substrate. The CogPrime perception process is intended to work perfectly well without access to the comprehensive database of experiences potentially stored in the ExperienceDB. However, a complete record of a mind's experience is a valuable thing, and there seems no reason for the system not to exploit it fully. Advantages like this allow the CogPrime system to partially compensate for its lack of some of the strengths of the human brain as an AI platform, such as massive parallelism.

26.3 Interfacing CogPrime with a Virtual Agent

We now discuss some of the particularities of connecting CogPrime to a virtual world (such as Second Life, Multiverse, or Unity3D, to name some of the virtual world / gaming platforms to which OpenCog has already been connected in practice).

26.3.1 Perceiving the Virtual World

The most complex, high-bandwidth sensory data coming in from a typical virtual world is visual data, so that will be our focus here. We consider three modes in which a virtual world may present visual data to CogPrime (or any other system):

- *Object vision*: CogPrime receives information about polygonal objects and their colors, textures and coordinates (each object is a set of contiguous polygons, and sometimes objects have "type" information, e.g. cube or sphere)
- *Polygon vision*: CogPrime receives information about polygons and their colors, textures and coordinates
- *Pixel vision*: CogPrime receives information about pixels and their colors and coordinates

In each case, coordinates may be given either in "world coordinates" or in "relative coordinates" (relative to the gaze). This distinction is not a huge deal since within an architecture like CogPrime , supplying schemata for

coordinate transformation is trivial; and, even if treated as a machine learning task, this sort of coordinate transformation is not very difficult to learn. Our current approach is to prefer relative coordinates, as this approach is more natural in terms of modern Western human psychology; but we note that in some other cultures world coordinates are preferred and considered more psychologically natural.

Currently we have not yet done any work with pixel vision in virtual worlds. We have been using object vision for most of our experiments, and consider a combination of polygon vision and object vision as the "right" approach for early AGI experiments in a virtual worlds context. The problem with pure object vision is that it removes the possibility for CogPrime to understand object segmentation. If, for instance, CogPrime perceives a person as a single object, then how can it recognize a head as a distinct sub-object? Feeding the system a pre-figured hierarchy of objects, sub-objects and so forth seems inappropriate in the context of an experiential learning system. On the other hand, the use of polygon vision instead of pixel vision seems to meet no such objections. This may take different forms in different platforms. For instance, in our work with a Minecraft-like world in the Unity3D environment, we have relied heavily on virtual objects made of blocks, in which case the polygons of most interest are the faces of the blocks.

Momentarily sticking with the object vision case for simplicity, examples of the perceptions emanating from the virtual world perceptual preprocessor into CogPrime are things like:

1. I am at world-coordinates $W
2. Object with metadata $M is at world-coordinates $W
3. Part of object with metadata $M is at world-coordinates $W
4. Avatar with metadata $M is at world-coordinates $W
5. Avatar with metadata $M is carrying out animation $A
6. Statements in natural language, from the pet owner

The perceptual preprocessor takes these signals and translates them into Atoms, making use of the special Atomspace mechanisms for efficiently indexing spatial and temporal information (the and) as appropriate.

26.3.1.1 Transforming Real-World Vision into Virtual Vision

One approach to enabling CogPrime to handle visual data coming from the real world is to transform this data into data of the type CogPrime sees in the virtual world. While this is not the approach we are taking in our current work, we do consider it a viable strategy, and we briefly describe it here.

One approach along these lines would involve multiple phases:

- Use a camera eye and a LiDAR (Light Detection And Ranging, used for high-resolution topographic mapping) sensor in tandem, so as to avoid having to deal with stereo vision

- Using the above two inputs, create a continuous 3D contour map of the perceived visual world
- Use standard mathematical transforms to polygon-ize the 3D contour map into a large set of small polygons
- Use heuristics to merge together the small polygons, obtaining a smaller set of larger polygons (but retaining the large set of small polygons for the system to reference in cases where a high level of detail is necessary)
- Feed the polygons into the perceptual pattern mining subsystem, analogously to the polygons that come in from virtual-world

In this approach, preprocessing is used to make the system see the physical world in a manner analogous to how it sees the virtual-world world. This is quite different from the DeSTIN-based approach to CogPrime vision that we will discuss in Chapter 28, but may well also be feasible.

26.3.2 Acting in the Virtual World

Complementing the perceptual preprocessor is the action postprocessor: code that translates the actions and action-sequences generated by CogPrime into instructions the virtual world can understand (such as "launch thus-and-thus animation"). Due to the particularities of current virtual world architectures, the current OpenCogPrime system carries out actions via executing preprogrammed high-level procedures, such as "move forward one step", "bend over forward" and so forth. Example action commands are:

1. Move ($D, $S) : $D is a distance, $S is a speed
2. Turn ($A, $S) : $A is an angle, $S is a speed
3. Pitch ($A, $S) : turn vertically up/down... [for birds only]
4. Jump ($D, $H, $S) : $H is a maximum height, at the center of the jump
5. Say ($T), $T is text : for agents with linguistic capability, which is not enabled in the current version
6. pick up($O) : $O is an object
7. put down($O)

This is admittedly a crude approach, and if a robot simulator rather than a typical virtual world were used, it would be possible for CogPrime to emanate detailed servomotor control commands rather than high-level instructions such as these. However, as noted in Chapter 16 of Part 1, at the moment there is no such thing as a "massive multiplayer robot simulator," and so the choice is between a multi-participant virtual environment (like the Multiverse environment currently used with the PetBrain) or a small-scale robot simulator. Our experiments with virtual worlds so far have used the high-level approach described here; but we are also experimenting with using physical robots and corresponding simulators, as will be described below.

26.4 Perceptual Pattern Mining

Next we describe how perceptual pattern mining may be carried out, to recognize meaningful structures in the stream of data produced via perceiving a virtual or physical world.

In this subsection we discuss the representation of knowledge, and then in the following subsection we discuss the actual mining. We discuss the process in the context of virtual-world perception as outlined above, but the same processes apply to robotic perception, whether one takes the "physical world as virtual world" approach described above or a different sort of approach such as the DeSTIN hybridization approach described below.

26.4.1 Input Data

First, we may assume that each perception is recorded as set of "transactions", each of which is of the form

```
Time, 3D coordinates, object type
```

or

```
Time, 3D coordinates, action type
```

Each transaction may also come with an additional list of (attribute, value) pairs, where the list of attributes is dependent upon the object or action type. Transactions are represented as Atoms, and don't need to be a specific Atom type - but are referred to here by the special name *transactions* simply to make the discussion clear.

Next, define a transaction template as a transaction with location and time information set to *wild cards* - and potentially, some other attributes set to wild cards. (These are implemented in terms of Atoms involving VariableNodes.)

For instance, some transaction templates in the current virtual-world might be informally represented as:

- Reward
- Red cube
- kick
- move_forward
- Cube
- Cube, size 5
- me
- Teacher

26.4.2 Transaction Graphs

Next we may conceive a transaction graph, whose nodes are transactions and whose links are labeled with labels like after, SimAND, SSeqAND (short for SimultaneousSequentialAND), near, in_front_of, and so forth (and whose links are weighted as well).

We may also conceive a transaction template graph, whose nodes are transaction templates, and whose links are the same as in the transaction graph. Examples of transaction template graphs are

```
near(Cube, Teacher)
```

```
SSeqAND(move_forward, Reward)
```

where Cube, Teacher, etc are transaction templates since Time and 3D coordinates are left unspecified.

And finally, we may conceive a transaction template relationship graph (TTRG), whose nodes may be any of: transactions; transaction templates; basic spatiotemporal predicates evaluated at tuples of transactions or transaction templates. For instance

```
SimAND(near(Cube, Teacher), above(Cube, Chair))
```

26.4.3 Spatiotemporal Conjunctions

Define a temporal conjunction as a conjunction involving SimultaneousAND and SequentialAND operators (including SSeqAND as a special case of SeqAND: the special case that interests us in the short term). The conjunction is therefore ordered, e.g.

```
A SSeqAND B SimAND C SSeqAND D
```

We may assume that the order of operations favors SimAND, so that no parenthesizing is necessary.

Next, define a basic spatiotemporal conjunction as a temporal conjunction that conjoins terms that are either

- transactions, or
- transaction templates, or
- basic spatiotemporal predicates applied to tuples of transactions or transaction templates

I.e. a basic spatiotemporal conjunction is a temporal conjunction of nodes from the transaction template relationship graph.

An example would be:

```
(hold ball) SimAND ( near(me, teacher) ) SSeqAND Reward
```

This assumes that the *hold* action has an attribute that is the type of object held, so that

```
hold ball
```

in the above temporal conjunction is a shorthand for the transaction template specified by

```
action type: hold

object_held_type: ball
```

This example says that if the agent is holding the ball and is near the teacher then shortly after that, the agent will get a reward.

26.4.4 The Mining Task

The perceptual mining task, then, is to find basic spatiotemporal conjunctions that are *interesting.* What constitutes interestingness is multifactorial, and includes.

- involves important Atoms (e.g. Reward)
- has a high temporal cohesion (i.e. the strength of the time relationships embodied in the SimAND and SeqAND links is high)
- has a high spatial cohesion (i.e. the near() relationships have high strength)
- has a high frequency
- has a high surprise value (its frequency is far from what would be predicted by its component sub-conjunctions)

Note that a conjunction can be interesting without satisfying all these criteria; e.g. if it involves something important and has a high temporal cohesion, we want to find it regardless of its spatial cohesion.

In preliminary experiments we have worked with a provisional definition of"interestingness" as the combination of frequency and temporal cohesion, but this must be extended; and one may even wish to have the combination function optimized over time (slowly) where the fitness function is defined in terms of the STI and LTI of the concepts generated.

26.4.4.1 A Mining Approach

One tractable approach to perceptual pattern mining is greedy and iterative, involving the following steps:

1. Build an initial transaction template graph G

2. Greedily mine some interesting basic spatiotemporal conjunctions from it, adding each interesting conjunction found as a new node in G (so that G becomes a transaction template relationship graph), repeating step 2 until boredom results or time runs out

The same TTRG may be maintained over time, but of course will require a robust forgetting mechanism once the history gets long or the environment gets nontrivially complex.

The greedy mining step may involve simply grabbing SeqAND or SimAND links with probability determined by the (importance and/or interestingness) of their targets, and the probabilistic strength and temporal strength of the temporal AND relationship, and then creating conjunctions based on these links (which then become new nodes in the TTRG, so they can be built up into larger conjunctions).

26.5 The Perceptual-Motor Hierarchy

The perceptual pattern mining approach described above is "flat," in the sense that it simply proposes to recognize patterns in a stream of perceptions, without imposing any kind of explicitly hierarchical structure on the pattern recognition process or the memory of perceptual patterns. This is different from how the human visual system works, with its clear hierarchical structure, and also different from many contemporary vision architectures, such as DeSTIN or Hawkins' Numenta system which also utilizes hierarchical neural networks.

However, the approach described above may be easily made hierarchical within the CogPrime architecture, and this is likely the most effective way to deal with complex visual scenes. Most simply, in this approach, a hierarchy may be constructed corresponding to different spatial regions, within the visual field. The RegionNodes at the lowest level of the hierarchy correspond to small spatial regions, the ones at the next level up correspond to slightly larger spatial regions, and so forth. Each RegionNode also correspond to a certain interval of time, and there may be different RegionNodes corresponding to the same spatial region but with different time-durations attached to them. RegionNodes may correspond to overlapping rather than disjoint regions.

Within each region mapped by a RegionNode, then, perceptual pattern mining as defined in the previous section may occur. The patterns recognized in a region are linked to the corresponding RegionNode - and are then fed as inputs to the RegionNodes corresponding to larger, encompassing regions; and as suggestions-to-guide-pattern-recognition to nearby RegionNodes on the same level. This architecture involves the fundamental hierarchical structure/dynamic observed in the human visual cortex. Thus, the hierarchy incurs

a dynamic of patterns-within-patterns-within-patterns, and the heterarchy incurs a dynamic of patterns-spawning-similar-patterns.

Also, patterns found in a RegionNode should be used to bias the pattern-search in the RegionNodes corresponding to smaller, contained regions: for instance, if many of the sub-regions corresponding to a certain region have revealed parts of a face, then the pattern-mining processes in the remaining sub-regions may be instructed to look for other face-parts.

This architecture permits the hierarchical dynamics utilized in standard hierarchical vision models, such as Jeff Hawkins' and other neural net models, but within the context of CogPrime 's pattern-mining approach to perception. It is a good example of the flexibility intrinsic to the CogPrime architecture.

Finally, why have we called it a perceptual-*motor* hierarchy above? This is because, due to the embedding of the perceptual hierarchy in CogPrime 's general Atom-network, the percepts in a certain region will automatically be linked to actions occurring in that region. So, there may be some perception-cognition-action interplay specific to a region, occurring in parallel with the dynamics in the hierarchy of multiple regions. Clearly this mirrors some of the complex dynamics occurring in the human brain, and is also reflected in the structure of sophisticated perceptual-motor approaches like DeSTIN, to be discussed below.

26.6 Object Recognition from Polygonal Meshes

Next we describe a more specific perceptual pattern recognition algorithm – a strategy for identifying objects in a visual scene that is perceived as a set of polygons. It is not a thoroughly detailed algorithmic approach, but rather a high-level description of how this may be done effectively within the CogPrime design. It is offered here largely as an illustration of how specialized perceptual data processing algorithms may be designed and implemented within the CogPrime framework.

We deal here with an agent whose perception of the world, at any point in time, is understood to consist of a set of polygons, each one described in terms of a list of corners. The corners may be assumed to be described in coordinates relative to the viewing eye of the agent.

What we mean by "identifying objects" here is something very simple. We don't mean identifying that a particular object is a chair, or is Ben's brown chair, or anything like that - we simply mean identifying that a given collection of polygons is meaningfully grouped into an object. That is the task considered here. The object could be a single block, it could be a person, or it could be a tower of blocks (which appears as a single object until it is taken apart).

Of course, not all approaches to polygon-based vision processing would require this sort of phase: it would be possible, as an alternative, to simply

compare the set of polygons in the visual field to a database of prior experience and then do object identification (in the present sense) based on this database-comparison. But in the approach described in this section, one begins instead with an automated segmentation of the set of perceived polygons into a set of objects.

26.6.1 Algorithm Overview

The algorithm described here falls into three stages:

1. Recognizing PersistentPolygonNodes (PPNodes) from PolygonNodes.
2. Creating Adjacency Graphs from PPNodes.
3. Clustering in the Adjacency Graph.

Each of these stages involves a bunch of details, not all of which have been fully resolved: this document just gives a conceptual overview.

We will speak in terms of objects such as PolygonNode, PPNode and so forth, because inside the CogPrime AI engine, observed and conceived entities are represented as nodes in an graph. However, this terminology is not very important here, and what we call a PolygonNode here could just as well be represented in a host of other ways, within the overall CogPrime framework.

26.6.2 Recognizing PersistentPolygonNodes (PPNodes) from PolygonNodes

A PolygonNode represents a polygon observed at a point in time. A PPNode represents a series of PolygonNodes that are heuristically guessed to represent the same PolygonNode at different moments in time.

Before "object permanence" is learned, the heuristics for recognizing PPNodes will only work in the case of a persistent polygon that, over an interval of time, is experiencing relative motion within the visual field, but is never leaving the visual field. For example some reasonable heuristics are: If P1 occurs at time t, P2 occurs at time s where s is very close to t, and P1 are similar in shape, size and color and position, then P1 and P2 should be grouped together into the same PPNode.

More advanced heuristics would deal carefully with the case where some of these similarities did not hold, which would allow us to deal e.g. with the case where an object was rapidly changing color.

In the case where the polygons are coming from a simulation world like OpenSim, then from our positions as programmers and world-masters, we can see that what a PPNode is supposed to correspond to is a certain side of a certain OpenSim object; but it doesn't appear immediately that way

to CogPrime when controlling an agent in OpenSim since CogPrime isn't perceiving OpenSim objects, it's perceiving polygons. On the other hand, in the case where polygons are coming from software that postprocesses the output of a LiDAR based vision system, then the piecing together of PPNodes from PolygonNodes is really necessary.

26.6.3 Creating Adjacency Graphs from PPNodes

Having identified PPNodes, we may then draw a graph between PPNodes, a PPGraph (also called an "Adjacency Graph"), wherein the links are AdjacencyLinks (with weights indicating the degree to which the two PPNodes tend to be adjacent, over time). A more refined graph might also involve SpatialCoordinationLinks (with weights indicating the degree to which the vector between the centroids of the two PPNodes tends to be consistent over time).

We may then use this graph to do object identification:

- First-level objects may be defined as clusters in the graph of PPNodes.
- One may also make a graph between first-level objects, an ObjectGraph with the same kinds of links as in the PPGraph. Second-level objects may be defined as clusters in the ObjectGraph.

The "strength" of an identified object may be assigned as the "quality" of the cluster (measured in terms of how tight the cluster is, and how well separated from other clusters.)

As an example, consider a robot with two parts: a body and a head. The whole body may have a moderate strength as a first-level object, but the head and body individually will have significantly greater strengths as first-level objects. On the other hand, the whole body should have a pretty strong strength as a second-level object.

It seems convenient (though not necessary) to have a PhysicalObjectNode type to represent the objects recognized via clustering; but the first versus second level object distinction should not need to be made on the Atom type level.

Building the adjacency graph requires a mathematical formula defining what it means for two PPNodes to be adjacent. Creating this formula may require a little tuning. For instance, the adjacency between two PPNodes PP1 and PP2 may be defined as the average over time of the adjacency of the PolygonNodes PP1(t) and PP2(t) observed at each time t. (A $p'th$ power average[1] may be used here, and different values of p may be tried.) Then, the adjacency between two (simultaneous) PolygonNodes P1 and P2 may be defined as the average over all x in P1 of the minimum over all y in P2 of sim(x,y), where sim(,) is an appropriately scaled similarity function. This

[1] the $p'th$ power average is defined as $\sqrt[p]{\sum X^p}$

latter average could arguably be made a maximum; or perhaps even better a $p'th$ power average with large p, which approximates a maximum.

26.6.4 Clustering in the Adjacency Graph.

As noted above, the idea is that objects correspond to clusters in the adjacency graph. This means we need to implement some hierarchical clustering algorithm that is tailored to find clusters in symmetric weighted graphs. Probably some decent algorithms of this character exist, if not it would be fairly easy to define one, e.g. by mapping some standard hierarchical clustering algorithm to deal with graphs rather than vectors.

Clusters will then be mapped into PhysicalObjectNodes, interlinked appropriately via PhysicalPartLinks and AdjacencyLinks. (E.g. there would be a PhysicalPartLink between the PhysicalObjectNode representing a head and the PhysicalObjectNode representing a body [where the body is considered as including the head]).

26.6.5 Discussion

It seems probable that, for simple scenes consisting of a small number of simple objects, clustering for object recognition will be fairly unproblematic. However, there are two cases that are potentially tricky:

- Sub-objects: e.g. the head and torso of a body, which may move separately; or the nose of the head, which may wiggle; or the legs of a walking dog; etc.
- Coordinated objects: e.g. if a character's hat is on a table, and then later on his head, then when it's on his head we basically want to consider him and his hat as the same object, for some purposes.

These examples show that partitioning a scene into *objects* is a borderline-cognitive rather than purely lower-level-perceptual task, which cannot be hard-wired in any very simple way.

We also note that, for complex scenes, clustering may not work perfectly for object recognition and some reasoning may be needed to aid with the process. Intuitively, these may correspond to scenes that, in human perceptual psychology, require conscious attention and focus in order to be accurately and usefully perceived.

26.7 Interfacing the Atomspace with a Deep Learning Based Perception-Action Hierarchy

We have discussed how one may do perception processing such as object recognition within the Atomspace, and this is indeed a viable strategy. But an alternate approach is also interesting, and likely more valuable in the case of robotic perception/action: build a separate perceptual-motor hierarchy, and link it in with the Atomspace. This approach is appealing in large part because a lot of valuable and successful work has already been done using neural networks and related architectures for perception and actuation. And it is not necessarily contradictory to doing perception processing in the Atomspace – obviously, one may have complementary, synergetic perception processing occurring in two different parts of the architecture.

This section reviews some general ideas regarding the interfacing of CogPrime with deep learning hierarchies for perception and action; the following chapter then discusses one example of this in detail, involving the DeSTIN deep learning architecture.

26.7.1 Hierarchical Perception Action Networks

CogPrime could be integrated with a variety of different hierarchical perception/action architectures. For the purpose of this section, however, we will consider a class of architectures that is neither completely general nor extremely specific. Many of the ideas to be presented here are in fact more broadly applicable beyond the architecture described here.

The following assumptions will be made about the HPANs (Hierarchical Perception/Action Network) to be hybridized with CogPrime . It may be best to use multiple HPANs, at least one for declarative/sensory/episodic knowledge (we'll call this the "primary HPAN") and one for procedural knowledge. A HPAN for intentional knowledge (a goal hierarchy; in DeSTIN called the "critic hierarchy") may be valuable as well. We assume that each HPAN has the properties:

1. It consists of a network of nodes, endowed with a learning algorithm, whose connectivity pattern is largely but not entirely hierarchical (and whose hierarchy contains both feedback, feedforward and lateral connections)
2. It contains a set of input nodes, receiving perceptual inputs, at the bottom of the hierarchy
3. It has a set of output nodes, which may span multiple levels of the hierarchy. The "output nodes" indicate informational signals to cognitive processes lying outside the HPAN, or else control signals to actuators, which may be internal or external.

4. Other nodes besides I/O nodes may potentially be observed or influenced by external processes; for instance they may receive stimulation
5. Link weights in the HPAN get updated via some learning algorithm that is roughly speaking "statistically Hebbian," in the sense that on the whole when a set of nodes get activated together for a period of time, they will tend to become attractors. By an attractor we mean: a set S of nodes such that the activation of a subset of S during a brief interval tends to lead to the activation of the whole set S during a reasonably brief interval to follow
6. As an approximate but not necessarily strict rule, nodes higher in the hierarchy tend to be involved in attractors corresponding to events or objects localized in larger spacetime regions

Examples of specific hierarchical architectures broadly satisfying these requirements are the visual pattern recognition networks constructed by Hawkins [**?**] and [PCP00], and Arel's DeSTIN system discussed earlier (and in more depth in following chapters). The latter appears to fit the requirements particularly snugly due to having dynamics very well suited to the formation of a complex array of attractors, and a richer methodology for producing outputs. These are all not only HPANs but have a more particular structure that in Chapter 27 is called a Compositional Spatiotemporal Deep Learning Network or CSDLN.

The particulars of the use of HPANs with OpenCog are perhaps best explained via enumeration of memory types and control operations.

26.7.2 Declarative Memory

The key idea here is linkage of primary HPAN attractors to CogPrime

ConceptNodes via MemberLinks. This is in accordance with the notion of glocal memory, in the language of which the HPAN attractors are the maps and the corresponding ConceptNodes are the keys. Put simply, when a HPAN attractor is recognized, MemberLinks are created between the HPAN nodes comprising the main body of the attractor, and a ConceptNode in the AtomTable representing the attractor. MemberLink weights may be used to denote fuzzy attractor membership. Activation may spread from HPAN nodes to ConceptNodes, and STI may spread from ConceptNodes to HPAN nodes; a conversion rate between HPAN activation and STI currency must be maintained by the CogPrime

central bank (see Chapter 23), for ECAN purposes.

Both abstract and concrete knowledge may be represented in this way. For instance, the Eiffel Tower would correspond to one attractor, the general shape of the Eiffel Tower would correspond to another, and the general notion of a "tower" would correspond to yet another. As these three examples

are increasingly abstract, the corresponding attractors would be weighted increasingly heavily on the upper levels of the hierarchy.

26.7.3 Sensory Memory

CogPrime may also use its primary HPAN to store memories of sense-perceptions and low-level abstractions therefrom. MemberLinks may join concepts in the AtomTable to percept-attractors in the HPAN. If the HPAN is engineered to associate specific neural modules to specific spatial regions or specific temporal intervals, then this may be accounted for by automatically indexing ConceptNodes corresponding to attractors centered in those modules in the AtomTable's TimeServer and SpaceServer objects, which index Atoms according to time and space.

An attractor representing something specific like the Eiffel Tower, or Bob's face, would be weighted largely in the lower levels of the hierarchy, and would correspond mainly to sensory rather than conceptual memory.

26.7.4 Procedural Memory

The procedural HPAN may be used to learn procedures such as low-level motion primitives that are more easily learned using HPAN training than using more abstract procedure learning methods. For example, a Combo tree learned by MOSES in CogPrime might contain a primitive corresponding to the predicate-argument relationship *pick_up(ball)*; but the actual procedure for controlling a robot hand to pick up a ball, might be expressed as an activity pattern within the low-level procedural HPAN. A procedure P stored in the low-level procedural HPAN would be represented in the AtomTable as a ConceptNode C linked to key nodes in the HPAN attractor corresponding to P. The invocation of P would be accomplished by transferring STI currency to C and then allowing ECAN to do its work.

On the other hand, CogPrime 's interfacing of the high-level procedural HPAN with the CogPrime ProcedureRepository is intimately dependent on the particulars of the MOSES procedure learning algorithm. As will be outlined in more depth in Chapter 33, MOSES is a complex, multi-stage process that tries to find a program maximizing some specified fitness function, and that involves doing the following within each "deme" (a deme being an island of roughly-similar programs)

1. casting program trees into a hierarchical normal form
2. evaluating the program trees on a fitness function

3. building a model distinguishing fit versus unfit program trees, which involves: 3a. figuring out what program tree features the model should include; 3b. building the model using a learning algorithm
4. generating new program trees that are inferred likely to give high fitness, based on the model
5. return to step 1 with these new program trees

There is also a system for managing the creation and deletion of demes.

The weakest point in CogPrime 's current MOSES-based approach to procedure learning appears to be step 3. And the main weakness is conceptual rather than algorithmic; what is needed is to replace the current step 3 with something that uses long-term memory to do model-building and feature-selection, rather than (like the current code) doing these things in a manner that's restricted to the population of program trees being evolved to optimize a particular fitness function.

One promising approach to resolving this issue is via replacing step 3b (and, to a limited extent, 3a) with an interconnection between MOSES and a procedural HPAN. A HPAN can do supervised categorization, and can be designed to handle feature selection in a manner integrated with categorization, and also to integrate long-term memory into its categorization decisions.

26.7.5 Episodic Memory

In a hybrid CogPrime /HPAN architecture, episodic knowledge may be handled via a combination of:

1. using a traditional approach to store a large ExperienceDB of actual experienced episodes [including sensory inputs and actions; and also the states of the most important items in memory during the experience]
2. using the Atomspace (with its TimeServer and SpaceServer components) to store declarative knowledge about experiences
3. using dimensional embedding to index the AtomSpace's episodic knowledge in a spatiotemporally savvy way, as described in Chapter 40
4. training a large HPAN to summarize the scope of experienced episodes (this could be the primary HPAN used for declarative and sensory memory, or could potentially be a separate episodic HPAN)

Such a network should be capable of generating imagined episodes based on cues, as well recalling real episodes. The HPAN would serve as a sort of index into the memory of episodes, There would be HebbianLinks from the AtomTable into the episodic HPAN.

For instance, suppose that once the agent built an extremely tall tower of blocks, taller than any others in its memory. Perhaps it wants to build another very tall tower again, so it wants to summon up the memory of that

previous occasion, to see if there is possibly guidance therein. It then proceeds by thinking about tallness and towerness at the same time, which stimulates the relevant episode, because at the time of building the extremely tall tower, the agent was thinking a lot about tallness (so thoughts of tallness are part of the episodic memory).

26.7.6 Action Selection and Attention Allocation

CogPrime 's action selection mechanism chooses procedures based on which ones are estimated most likely to achieve current goals given current context, and places these in an "active procedure pool" where an ExecutionManager object mediates their execution.

Attention allocation spans all components of CogPrime , including an HPAN if one is integrated. Attention flows between the two components due to the conversion of STI to and from HPAN activation. And in this manner assignment of credit flows from GoalNodes into the HPAN, because this kind of simultaneous activation may be viewed as "rewarding" a HPAN link. So, the HPAN may reward signals from GoalNodes via ECAN, because when a ConceptNode gets rewarded, if the ConceptNode points to a set of nodes, these nodes get some of the reward.

26.8 Multiple Interaction Channels

Now we discuss a broader issue regarding the interfacing between CogPrime and the external world. The only currently existing embodied OpenCog applications, PetBrain and CogBot, are based on a loosely human model of perception and action, in which a single CogPrime instance controls a single mobile body, but this of course is not the only way to do things. More generally, what we can say is that a variety of external-world events come into a CogPrime system from physical or virtual world sensors, plus from other sources such as database interfaces, Web spiders, and/or other sources. The external systems providing CogPrime with data may be generically referred to as *sensory sources* (and in the terminology we adopt here, once Atoms have been created to represent external data, then one is dealing with *perceptions* rather than sensations). The question arises how to architect a CogPrime system, in general, for dealing with a variety of sensory sources.

We introduce the notion of an "interaction channel": a collection of sensory sources that is intended to be considered as a whole as a synchronous stream, and that is also able to receive CogPrime actions - in the sense that when CogPrime carries out actions relative to the interaction channel, this directly affects the perceptions that CogPrime receives from the interaction

channel. A CogPrime meant to have conversations with 10 separate users at once might have 10 interaction channels. A human mind has only one interaction channel in this sense (although humans may become moderately adept at processing information from multiple external-world sources, coming in through the same interaction channel).

Multiple-interaction-channel digital psychology may become extremely complex - and hard for us, with our single interaction channels, to comprehend. This is one among many cases where a digital mind, with its more flexible architecture, will have a clear advantage over our human minds with their fixed and limited neural architectures. For simplicity, however, in the following chapters we will often focus on the single-interaction-channel case.

Events coming in through an interaction channel are presented to the system as new perceptual Atoms, and relationships amongst these. In the multiple interaction channel case, the AttentionValues of these newly created Atoms require special treatment. Not only do they require special rules, they require additional fields to be added to the AttentionValue object, beyond what has been discussed so far.

We require newly created perceptual Atoms to be given a high initial STI. And we also require them to be given a high amount of a quantity called "interaction-channel STI." To support this, the AttentionValue objects of Atoms must be expanded to contain interaction-channel STI values; and the ImportanceUpdating MindAgent must compute interaction-channel importance separately from ordinary importance.

And, just as we have channel-specific AttentionValues, we may also have channel-specific TruthValues. This allows the system to separately account for the frequency of a given perceptual item in a given interaction channel. However, no specific mechanism is needed for these, they are merely contextual truth values, to be interpreted within a Context Node associated with the interaction channel.

Chapter 27
Integrating CogPrime with a Compositional Spatiotemporal Deep Learning Network

27.1 Introduction

Many different approaches to "low-level" perception and action processing are possible within the overall CogPrime framework. We discussed several in the previous chapter, all elaborations of the general hierarchical pattern recognition approach. Here we describe one sophisticated approach to hierarchical pattern recognition based perception in more detail: the tight integration of CogPrime with a sophisticated hierarchical perception/action oriented learning system such as the DeSTIN architecture reviewed in Chapter 4 of Part 1.

We introduce here the term "Compositional Spatiotemporal Deep Learning Network" (CSDLN), to refer to deep learning networks whose hierarchical structure directly mirrors the hierarchical structure of spacetime. In the language of Chapter 26, a CSDLN is a special kind of HPAN (hierarchical perception action network), which has the special property that each of its nodes refers to a certain spatiotemporal region and is concerned with predicting what happens inside that region. Current exemplifications of the CSDLN paradigm include the DeSTIN architecture that we will focus on here, along with Jeff Hawkins' Numenta "HTM" system [HB06] [1], Itamar Arel's DeSTIN [ARC09], Itamar Arel's HDRN [2] system (the proprietary, closed-source sibling of DeSTIN), Dileep George's spin-off from Numenta [3], and work by Mohamad Tarifi [TSH11], Bundzel and Hashimoto [BH10], and others. CSDLNs are reasonably well proven as an approach to intelligent sensory data processing, and have also been hypothesized as a broader foundation for artificial general intelligence at the human level and beyond [HB06] [ARC09].

[1] While the Numenta system is the best-known CSDLN architecture, other CSDLNs appear more impressively functional in various respects; and many CSDLN-related ideas existed in the literature well before Numenta's advent.

[2] http:\binatix.com

[3] http:\vicarioussystems.com

While CSDLNs have been discussed largely in the context of perception, the specific form of CSDLN we will pursue here goes beyond perception processing, and involves the coupling of three separate hierarchies, for perception, action and goals/reinforcement [GLdG+10]. The "action CSDLNs discussed here correspond to the procedural HPAN discussed in Chapter 26. Abstract learning and self-understanding are then hypothesized as related to systems of attractors emerging from the close dynamic coupling of the upper levels of the three hierarchies. DeSTIN is our paradigm case of this sort of CSDLN, but most of the considerations given here would apply to any CSDLN of this general character.

CSDLNs embody a certain conceptual model of the nature of intelligence, and to integrate them appropriately with a broader architecture, one must perform the integration not only on the level of software code but also on the level of conceptual models. Here we focus here on the problem of integrating an extended version of the DeSTIN CSDLN system with the CogPrime integrative AGI (artificial general intelligence) system. The crux of the issue here is how to map DeSTIN's attractors into CogPrime 's more abstract, probabilistic "weighted, labeled hypergraph" representation (called the Atomspace). The main conclusion reached is that in order to perform this mapping in a conceptually satisfactory way, one requires a system of hierarchies involving **the structure of DeSTIN's network but the semantic structures of the Atomspace**. The DeSTIN perceptual hierarchy is augmented by motor and goal hierarchies, leading to a tripartite "extended DeSTIN". In this spirit, three "semantic-perceptual" hierarchies are proposed, corresponding to the three extended-DeSTIN CSDLN hierarchies and explicitly constituting an intermediate level of representation between attractors in DeSTIN and the habitual cognitive usage of CogPrime Atoms and Atomnetworks. For simple reference we refer to this as the "Semantic CSDLN" approach.

A "tripartite semantic CSDLN" consisting of interlinked semantic perceptual, motoric and goal hierarchies could be coupled with DeSTIN or another CSDLN architecture to form a novel AGI approach; or (our main focus here) it may be used as a glue between an CSDLN and and a more abstract semantic network such as the cognitive Atoms in CogPrime 's Atomspace.

One of the core intuitions underlying this integration is that, in order to achieve the desired level of functionality for tasks like picture interpretation and assembly of complex block structures, a convenient route is to perform a fairly *tight* integration a highly capable CSDLN like DeSTIN with other CogPrime components. For instance, we believe it's necessary to go deeper than just using DeSTIN as an input/output layer for CogPrime , by building associative links between the nodes inside DeSTIN and those inside the Atomspace.

This "tightly linked integration" approach is obviously an instantiation of the general cognitive synergy principle, which hypothesizes particular properties that the interactions between components in an integrated AGI system

should display, in order for the overall system to display significant general intelligence using limited computational resources. Simply piping output from an CSDLN to other components, and issuing control signals from these components to the CSDLN, is likely an inadequate mode of integration, incapable of leveraging the full potential of CSDLNs; what we are suggesting here is a much tighter and more synergetic integration.

In terms of the general principle of mind-world correspondence, the conceptual justification for CSDLN/CogPrime integration would be that the everyday human world contains many compositional spatiotemporal structures relevant to human goals, but also contains many relevant patterns that are not most conveniently cast into a compositional spatiotemporal hierarchy. Thus, in order to most effectively perceive, remember, represent, manipulate and enact the full variety of relevant patterns in the world, it is sensible to have a cognitive structure containing a CSDLN as a significant component, but not the only component.

27.2 Integrating CSDLNs with Other AI Frameworks

CSDLNs represent knowledge as attractor patterns spanning multiple levels of hierarchical networks, supported by nonlinear dynamics and (at least in the case of the overall DeSTIN design) involving cooperative activity of perceptual, motor and control networks. These attractors are learned and adapted via a combination of methods including localized pattern recognition algorithms and probabilistic inference. Other AGI paradigms represent and learn knowledge in a host of other ways. How then can CSDLNs be integrated with these other paradigms?

A very simple form of integration, obviously, would be to use an CSDLN as a sensorimotor cortex for another AI system that's focused on more abstract cognition. In this approach, the CSDLN would stream state-vectors to the abstract cognitive system, and the abstract cognitive system would stream abstract cognitive inputs to the CSDLN (which would then consider them together with its other inputs). One thing missing in this approach is the possibility of the abstract cognitive system's insights biasing the judgments inside the CSDLN. Also, abstract cognition systems aren't usually well prepared to handle a stream of quantitative state vectors (even ones representing intelligent compressions of raw data).

An alternate approach is to build a richer intermediate layer, which in effect translates between the internal language of the CSDLN and the internal language of the other AI system involved. The particulars, and the viability, of this will depend on the particulars of the other AI system. What we'll consider here is the case where the other AI system contains explicit symbolic representations of patterns (including patterns abstracted from observations that may have no relation to its prior knowledge or any linguistic terms).

In this case, we suggest, a viable approach may be to construct a "semantic CSDLN" to serve as an intermediary. The semantic CSDLN has the same hierarchical structure as an CSDLN, but inside each node it contains abstract patterns rather than numerical vectors. This approach has several potential major advantages: the other AI system is not presented with a large volume of numerical vectors (which it may be unprepared to deal with effectively); the CSDLN can be guided by the other AI system, without needing to understand symbolic control signals; and the intermediary semantic CSDLN can serve as a sort of "blackboard" which the CSDLN and the other AI system can update in parallel, and be guided by in parallel, thus providing a platform encouraging "cognitive synergy".

The following sections go into more detail on the concept of semantic CSDLNs. The discussion mainly concerns the specific context of DeSTIN/CogPrime integration, but the core ideas would apply to the integration of any CSDLN architecture with any other AI architecture involving uncertain symbolic representations susceptible to online learning.

27.3 Semantic CSDLN for Perception Processing

In the standard perceptual CSDLN hierarchy, a node N on level k (considering level 1 as the bottom) corresponds to a spatiotemporal region S with size s_k (s_k increasing monotonically and usually exponentially with k); and, has children on level $k-1$ corresponding to spatiotemporal regions that collectively partition S. For example, a node on level 3 might correspond to a $16x16$ pixel region S of 2D space over a time period of 10 seconds, and might have 4 level 2 children corresponding to disjoint $4x4$ regions of 2D space over 10 seconds, collectively composing S.

This kind of hierarchy is very effective for recognizing certain types of visual patterns. However it is cumbersome for recognizing some other types of patterns, e.g. the pattern that a face typically contains two eyes beside each other, but at variable distance from each other.

One way to remedy this deficiency is to extend the definition of the hierarchy, so that nodes do not refer to fixed spatial or temporal positions, but only to *relative* positions. In this approach, the internals of a node are basically the same as in an CSDLN, and the correspondence of the nodes on level k with regions of size s_k is retained, but the relationships between the nodes are quite different. For instance, a variable-position node of this sort could contain several possible 2D pictures of an eye, but be nonspecific about where the eye is located in the 2D input image.

Figure 27.1 depicts this "semantic-perceptual CSDLN" idea heuristically, showing part of a semantic-perceptual CSDLN indicating the parts of a face, and also the connections between the semantic-perceptual CSDLN, a stan-

dard perceptual CSDLN, and a higher-level cognitive semantic network like CogPrime 's Atomspace. [4]

More formally, in the suggested "semantic-perceptual CSDLN" approach, a node N on level k, instead of pointing to a set of level $k-1$ children, points to a small (but not necessarily connected) *semantic network* , such that the nodes of the semantic network are (variable-position) level $k-1$ nodes; and the edges of the semantic network possess labels representing spatial or temporal relationships, for example *horizontally_ aligned*, *vertically_ aligned*, *right_ side*, *left_ side*, *above*, *behind*, *immediately_ right*, *immediately_ left*, *immediately_ above*, *immediately_ below*, *after*, *immediately_ after*. The edges may also be weighted either with numbers or probability distributions, indicating the quantitative weight of the relationship indicated by the label.

So for example, a level 3 node could have a child network of the form horizontally_aligned(N_1, N_2) where N_1 and N_2 are variable-position level 2 nodes. This would mean that N_1 and N_2 are along the same horizontal axis in the 2D input but don't need to be immediately next to each other. Or one could say, e.g. on_axis_perpendicular_to(N_1, N_2, N_3, N_4), meaning that N_1 and N_2 are on an axis perpendicular to the axis between N_3 and N_4. It may be that the latter sort of relationship is fundamentally better in some cases, because *horizontally_aligned* is still tied to a specific orientation in an absolute space, whereas *on_axis_perpendicular_to* is fully relative. But it may be that both sorts of relationship are useful.

Next, development of learning algorithms for semantic CSDLNs seems a tractable research area. First of all, it would seem that, for instance, the DeSTIN learning algorithms could straightforwardly be utilized in the semantic CSDLN case, once the local semantic networks involved in the network are known. So at least for some CSDLN designs, the problem of learning the semantic networks may be decoupled somewhat from the learning occurring inside the nodes. DeSTIN nodes deal with clustering of their inputs, and calculation of probabilities based on these clusters (and based on the parent node states). The difference between the semantic CSDLN and the traditional DeSTIN CSDLN has to do with what the inputs are.

Regarding learning the local semantic networks, one relatively straightforward approach would be to *data mine them from a standard CSDLN*. That is, if one runs a standard CSDLN on a stream of inputs, one can then run a frequent pattern mining algorithm to find semantic networks (using a given vocabulary of semantic relationships) that occur frequently in the CSDLN as

[4] The perceptual CSDLN shown is unrealistically small for complex vision processing (only 4 layers), and only a fragment of the semantic-perceptual CSDLN is shown (a node corresponding to the category face, and then a child network containing nodes corresponding to several components of a typical face). In a real semantic-perceptual CSDLN, there would be many other nodes on the same level as the face node, many other parts to the face subnetwork besides the eyes, nose and mouth depicted here; the eye, nose and mouth nodes would also have child subnetworks; there would be link from each semantic node to centroids within a large number of perceptual nodes; and there would also be many nodes not corresponding clearly to any single English language concept like eye, nose, face, etc.

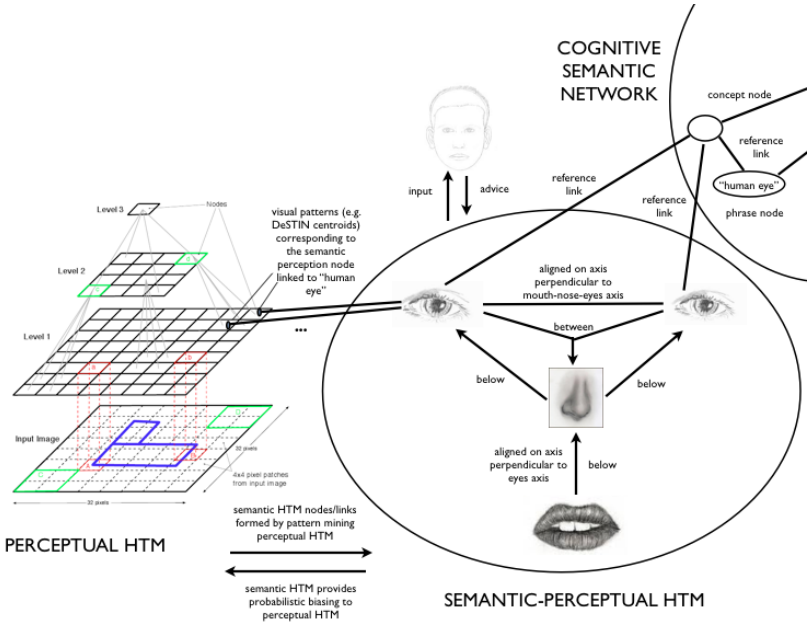


Fig. 27.1 Simplified depiction of the relationship between a semantic-perceptual CSDLN, a traditional perceptual CSDLN (like DeSTIN), and a cognitive semantic network (like CogPrime 's AtomSpace).

it processes input. A subnetwork that is identified via this sort of mining, can then be grouped together in the semantic CSDLN, and a parent node can be created and pointed to it.

Also, the standard CSDLN can be searched for frequent patterns involving the clusters (referring to DeSTIN here, where the nodes contain clusters of input sequences) inside the nodes in the semantic CSDLN. Thus, in the "semantic DeSTIN" case, we have a feedback interaction wherein: 1) the standard CSDLN is formed via processing input; 2) frequent pattern mining on the standard CSDLN is used to create subnetworks and corresponding parent nodes in the semantic CSDLN; 3) the newly created nodes in the semantic CSDLN get their internal clusters updated via standard DeSTIN dynamics; 4) the clusters in the semantic nodes are used as seeds for frequent pattern mining on the standard CSDLN, returning us to Step 2 above.

After the semantic CSDLN is formed via mining the perceptual CSDLN, it may be used to bias the further processing of the perceptual CSDLN. For instance, in DeSTIN each node carries out probabilistic calculations involving knowledge of the prior probability of the "observation" coming into that node over a given interval of time. In the current DeSTIN version, this prior probability is drawn from a uniform distribution, but it would be more effective to draw the prior probability from the semantic network – observa-

tions matching things represented in the semantic network would get a higher prior probability. One could also use subtler strategies such as using imprecise probabilities in DeSTIN [Goe11b], and assigning a greater confidence to probabilities involving observations contained in the semantic network.

Finally, we note that the nodes and networks in the semantic CSDLN may either

- be linked into the nodes and links in a semantic network such as CogPrime 's AtomSpace
- actually be implemented in terms of an abstract semantic network language like CogPrime 's AtomSpace (the strategy to be suggested in Chapter 29).

This allows us to think of the semantic CSDLN as a kind of bridge between the standard CSDLN and the cognitive layer of an AI system. In an advanced implementation, the cognitive network may be used to suggest new relationships between nodes in the semantic CSDLN, based on knowledge gained via inference or language.

27.4 Semantic CSDLN for Motor and Sensorimotor Processing

Next we consider a semantic CSDLN that focuses on movement rather than sensation. In this case, rather than a 2D or 3D visual space, one is dealing with an n-dimensional *configuration space* (C-space). This space has one dimension for each degree of freedom of the agent in question. The more joints with more freedom of movement an agent has, the higher the dimensionality of its configuration space.

Using the notion of configuration space, one can construct a *semantic-motoric CSDLN hierarchy* analogous to the semantic-perceptual CSDLN hierarchy. However, the curse of dimensionality demands a thoughtful approach here. A square of side 2 can be tiled with 4 squares of side 1, but a 50-dimensional cube of side 2 can be tiled with 2^{50} 50-dimensional cubes of side 1. If one is to build a CSDLN hierarchy in configuration space analogous to that in perceptual space, some sort of sparse hierarchy is necessary.

There are many ways to build a sparse hierarchy of this nature, but one simple approach is to build a hierarchy where the nodes on level k represent motions that combine the motions represented by nodes on level $k-1$. In this case the most natural semantic label predicates would seem to be things like *simultaneously*, *after*, *immediately_after*, etc. So a level k node represents a sort of "motion plan" corresponded by chaining together (serially and/or in parallel) the motions encoded in level $k-1$ nodes. Overlapping regions of C-space correspond to different complex movements that share some of the same component movements, e.g. if one is trying to slap one person while

elbowing another, or run while kicking a soccer ball forwards. Also note, the semantic CSDLN approach reveals perception and motor control to have essentially similar hierarchical structures, more so than with the traditional CSDLN approach and its fixed-position perceptual nodes.

Just as the semantic-perceptual CSDLN is naturally aligned with a traditional perceptual CSDLN, similarly a semantic-motoric CSDLN may be naturally aligned with a "motor CSDLN". A typical motoric hierarchy in robotics might contain a node corresponding to a robot arm, with children corresponding to the hand, upper arm and lower arm; the hand node might then contain child nodes corresponding to each finger, etc. This sort of hierarchy is intrinsically spatiotemporal because each individual action of each joint of an actuator like an arm is intrinsically bounded in space and time. Perhaps the most ambitious attempt along these lines is [AM01], which shows how perceptual and motoric hierarchies are constructed and aligned in an architecture for intelligent automated vehicle control.

Figure 27.2 gives a simplified illustration of the potential alignment between a semantic-motoric CSDLN and a purely motoric hierarchy (like the one posited above in the context of extended DeSTIN). [5] In the figure, the motoric hierarchy is assumed to operate somewhat like DeSTIN, with nodes corresponding to (at the lowest level) individual servomotors, and (on higher levels) natural groupings of servomotors. The node corresponding to a set of servos is assumed to contain centroids of clusters of trajectories through configuration space. The task of choosing an appropriate action is then executed by finding the appropriate centroids for the nodes. Note an asymmetry between perception and action here. In perception the basic flow is bottom-up, with top-down flow used for modulation and for "imaginative" generation of percepts. In action, the basic flow is top-down, with bottom-up flow used for modulation and for imaginative, "fiddling around" style generation of actions. The semantic-motoric hierarchy then contains abstractions of the C-space centroids from the motoric hierarchy – i.e., actions that bind together different C-space trajectories that correspond to the same fundamental action carried out in different contexts or under different constraints. Similarly to in the perceptual case, the semantic hierarchy here serves as a glue between lower-level function and higher-level cognitive semantics.

[5] In the figure, only a fragment of the semantic-motoric CSDLN is shown (a node corresponding to the "get object" action category, and then a child network containing nodes corresponding to several components of the action). In a real semantic-motoric CSDLN, there would be many other nodes on the same level as the get-object node, many other parts to the get-object subnetwork besides the ones depicted here; the subnetwork nodes would also have child subnetworks; there would be link from each semantic node to centroids within a large number of motoric nodes; and there might also be many nodes not corresponding clearly to any single English language concept like "grasp object" etc.

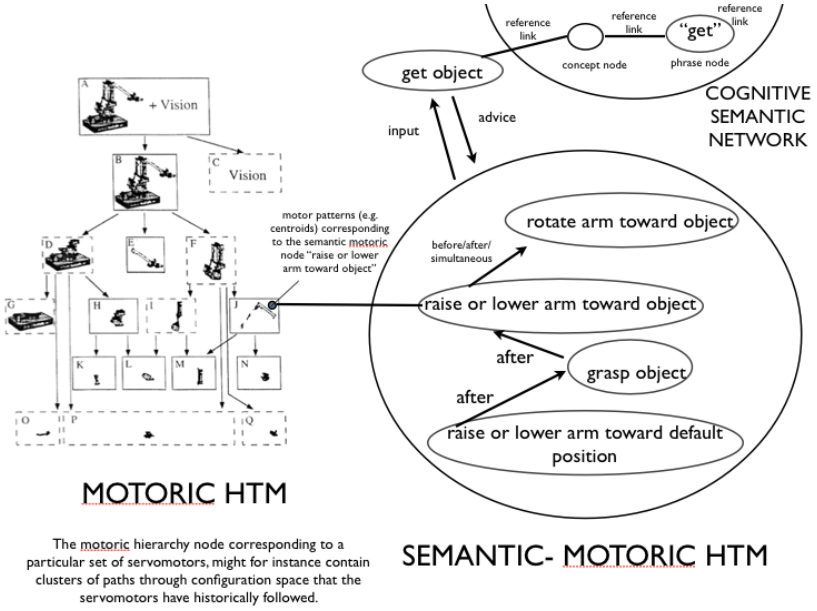


Fig. 27.2 Simplified depiction of the relationship between a semantic-motoric CSDLN, a motor control hierarchy (illustrated by the hierarchy of servos associated with a robot arm), and a cognitive semantic network (like CogPrime 's AtomSpace).

27.5 Connecting the Perceptual and Motoric Hierarchies with a Goal Hierarchy

One way to connect perceptual and motoric CSDLN hierarchies is using a "semantic-goal CSDLN" bridging the semantic-perceptual and semantic-motoric CSDLNs. The semantic-goal CSDLN would be a "semantic CSDLN" loosely analogous to the perceptual and motor semantic CSDLNs – and could optionally be linked into the reinforcement hierarchy of a tripartite CSDLN like extended DeSTIN. Each node in the semantic-goal CSDLN would contain implications of the form "Context & Procedure → Goal", where Goal is one of the AI system's overall goals or a subgoal thereof, and Context and Procedure refer to nodes in the perceptual and motoric semantic CSDLNs respectively.

For instance, a semantic-goal CSDLN node might contain an implication of the form "I perceive my hand is near object X & I grasp object X → I possess object X." This would be useful if "I possess object X" were a subgoal of some higher-level system goal, e.g. if X were a food object and the system had the higher-level goal of obtaining food.

To the extent that the system's goals can be decomposed into hierarchies of progressively more and more spatiotemporally localized subgoals, this sort of

hierarchy will make sense, leading to a tripartite hierarchy as loosely depicted in Figure 27.3. [6] One could attempt to construct an overall AGI approach based on a tripartite hierarchy of this nature, counting on the upper levels of the three hierarchies to come together dynamically to form an integrated cognitive network, yielding abstract phenomena like language, self, reasoning and mathematics. On the other hand, one may view this sort of hierarchy as a portion of a larger integrative AGI architecture, containing a separate cognitive network, with a less rigidly hierarchical structure and less of a tie to the spatiotemporal structure of physical reality. The latter view is the one we are primarily taking within the CogPrime AGI approach, viewing perceptual, motoric and goal hierarchies as "lower level" subsystems connected to a "higher level" system based on the CogPrime AtomSpace and centered on its abstract cognitive processes.

Learning of the subgoals and implications in the goal hierarchy is of course a complex matter, which may be addressed via a variety of algorithms, including online clustering (for subgoals or implications) or supervised learning (for implications, the "supervision" being purely internal and provided by goal or subgoal achievement).

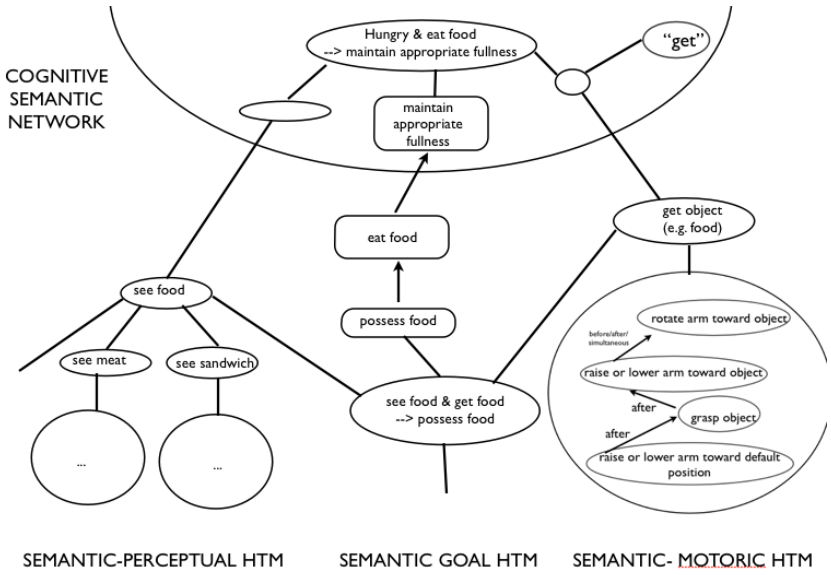


Fig. 27.3 Simplified illustration of the proposed interoperation of perceptual, motoric and goal semantic CSDLNs.

[6] The diagram is simplified in many ways, e.g. only a handful of nodes in each hierarchy is shown (rather than the whole hierarchy), and lines without arrows are used to indicate bidirectional arrows, and nearly all links are omitted. The purpose is just to show the general character of interaction between the components in a simplified context.

Chapter 28
Making DeSTIN Representationally Transparent

Co-authored with Itamar Arel

28.1 Introduction

In this chapter and the next we describe one particular incarnation of the above ideas on semantic CSDLNs in more depth: the integration of CogPrime with the DeSTIN architecture reviewed in Chapter 4 of Part 1.

One of the core intuitions underlying this integration is that, in order to achieve the desired level of functionality for tasks like picture interpretation and assembly of complex block structures, it will be necessary to integrate DeSTIN (or some similar system) and CogPrime components fairly tightly – going deeper than just using DeSTIN as an input/output layer for CogPrime , by building a number of explicit linkages between the nodes inside DeSTIN and CogPrime respectively.

The general DeSTIN design has been described in talks as comprising three crosslinked hierarchies, handling perception, action and reinforcement; but so far only the perceptual hierarchy (also called the "spatiotemporal inference network") has been implemented or described in detail in publications. In this chapter we will focus on DeSTIN's perception hierarchy. We will explain DeSTIN's perceptual dynamics and representations as we understand them, more thoroughly than was done in the brief review above; and we will describe a series of changes to the DeSTIN design, made in the spirit of easing DeSTIN/OpenCog integration. In the following chapter we will draw action and reinforcement into the picture, deviating somewhat in the details from the manner in which these things would be incorporated into a standalone DeSTIN, but pursuing the same concepts in an OpenCog integration context.

What we describe here is a way to make a "Uniform DeSTIN", in which the internal representation of perceived visual forms is independent of affine transformations (translation, scaling, rotation and shear). This "representational transparency" means that, when Uniform DeSTIN perceives a pattern: no matter how that pattern is shifted or linearly transformed, the way Uniform DeSTIN represents that pattern internally is going to be basically the

same. This makes it easy to look at a collection of DeSTIN states, obtained by exposing a DeSTIN perception network to the world at different points in time, and see the commonalities in what they are perceiving and how they are interpreting it. By contrast, in the original version of DeSTIN (here called "classic DeSTIN"), it may take significant effort to connect the internal representation of a visual pattern and the representation of its translated or linearly transformed versions. The uniformity of Uniform DeSTIN makes it easier for humans to inspect DeSTIN's state and understand what's going on, and also (more to the point) makes it easier for other AI components to recognize patterns in sets of DeSTIN states. The latter fact is critical for the DeSTIN/OpenCog integration

28.2 Review of DeSTIN Architecture and Dynamics

The hierarchical architecture of DeSTIN's spatiotemporal inference network comprises an arrangement into multiple layers of "nodes" comprising multiple instantiations of an identical processing unit. Each node corresponds to a particular spatiotemporal region, and uses a statistical learning algorithm to characterize the sequences of patterns that are presented to it by nodes in the layer beneath it.

More specifically, at the very **lowest layer** of the hierarchy nodes receive as input raw data (e.g. pixels of an image) and continuously construct a belief state that attempts to characterize the sequences of patterns viewed. The **second layer**, and all those above it, receive as input the belief states of nodes at their corresponding lower layers, and attempt to construct belief states that capture regularities in their inputs. Each node also receives as input the belief state of the node above it in the hierarchy (which constitutes "contextual" information, utilized in the node's prediction process).

Inside each node, an online clustering algorithm is used to identify regularities in the sequences received by that node. The centroids of the clusters learned are stored in the node and comprise the basic visual patterns recognized by that node. The node's "belief" regarding what it is seeing, is then understood as a probability density function defined over the centroids at that node. The equations underlying this centroid formation and belief updating process are identical for every node in the architecture, and were given in their original form in [ARC09], though the current open-source DeSTIN codebase reflects some significant improvements not yet reflected in the publication record.

In short, the way DeSTIN represents an item of knowledge is as a probability distribution over "network activity patterns" in its hierarchical network. An activity pattern, at each point in time, comprises an indication of which centroids in each node are most active, meaning they have been identified as most closely resembling what that node has perceived, as judged in the

context of the perceptions of the other nodes in the system. Based on this methodology, the DeSTIN perceptual network serves the critical role of building and maintaining a model of the state of the world as visually perceived.

This methodology allows for powerful unsupervised classification. If shown a variety of real-world scenes, DeSTIN will automatically form internal structures corresponding to the various natural categories of objects shown in the scenes, such as trees, chairs, people, etc.; and also to the various natural categories of events it sees, such as reaching, pointing, falling. In order to demonstrate the informativeness of these internal structures, experiments have been done using DeSTIN's states as input feature vectors for supervised learning algorithms, enabling high-accuracy supervised learning of classification models from labeled image data [KAR10]. A closely related algorithm developed by the same principal researcher (Itamar Arel) has proven extremely successful at audition tasks such as phoneme recognition [ABS+11].

28.2.1 Beyond Gray-Scale Vision

The DeSTIN approach may easily be extended to other senses beyond gray-scale vision. For color vision, it suffices to replace the one-dimensional signals coming into DeSTIN's lower layer with 3D signals representing points in the color spectrum; the rest of the DeSTIN process may be carried over essentially without modification. Extension to further senses is also relatively straightforward on the mathematical and software structure level, though they may of course require significant additional tuning and refinement of details.

For instance, while olfaction does not lend itself well to hierarchical modeling, but audition and haptics (touch) do:

- for auditory perception, one could use a DeSTIN architecture in which each layer is one-dimensional rather than two-dimensional, representing a certain pitch. Or one could use two dimensions for pitch and volume. This results in a system quite similar to the DeSTIN-like system shown to perform outstanding phoneme recognition in [ABS+11], and is conceptually similar to Hierarchical Hidden Markov Models (HHMMs), which have proven quite successful in speech recognition and which Ray Kurzweil has argued are the central mechanism of human intelligence [Kur12]. Note also recent results published by Microsoft Research, showing dramatic improvements over prior speech recognition results based on use of a broadly HHMM-like deep learning system [HDY+12].
- for haptic perception, one could use a DeSTIN architecture in which the lower layer of the network possesses a 2D topology reflecting the topology of the surface of the body. Similar to the somatosensory cortex in the human brain, the map could be distorted so that more "pixels" are used for regions of the body from which more data is available (e.g. currently

this might be the fingertips, if these were implemented using Syntouch technology [FL12], which has proved excellent at touch-based object identification). Input could potentially be multidimensional if multiple kinds of haptic sensors were available, e.g temperature, pressure and movement as in the Syntouch case.

Augmentation of DeSTIN to handle action as well as perception is also possible, and will be discussed in Chapter 29

28.3 Uniform DeSTIN

It would be possible to integrate DeSTIN in its original form with OpenCog or other AI systems with symbolic aspects, via using an unsupervised machine learning algorithm to recognize patterns in sets of states of the DeSTIN network as originally defined. However, this pattern recognition task becomes much easier if one suitably modifies DeSTIN, so as to make the commonalities between semantically similar states more obviously perceptible. This can be done by making the library of patterns recognized within each DeSTIN node invariant with respect to translation, scale, rotation and shear – a modification we call "Uniform DeSTIN." This "uniformization" decreases DeSTIN's degree of biological mimicry, but eases integration of DeSTIN with symbolic AI methods.

28.3.1 Translation-Invariant DeSTIN

The first revision to the "classic DeSTIN" to be suggested here is: All the nodes on the same level of the DeSTIN hierarchy should share the same library of patterns. In the context of classic DeSTIN (i.e. in the absence of further changes to DeSTIN to be suggested below, which extend the type of patterns usable by DeSTIN), this means: the nodes on the same level should share the same list of centroids. This makes DeSTIN's pattern recognition capability translation-invariant. This translation invariance can be achieved without any change to the algorithms for updating centroids and matching inputs to centroids.

In this approach, it's computationally feasible to have a much larger library of patterns utilized by each node, as compared to in classic DeSTIN. Suppose we have a $n \times n$ pixel grid, where the lowest level has nodes corresponding to 4×4 squares. Then, there are $(\frac{n}{4})^2$ nodes on the lowest level, and on the k'th level there are $(\frac{n}{4^k})^2$ nodes. This means that, without increasing computational complexity (actually decreasing it, under reasonable assumptions), in translation-invariant Uniform DeSTIN we can have a factor of $(\frac{n}{4^k})^2$ more centroids on level k.

One can achieve a much greater decrease in computational complexity (with the same amount of centroid increase) via use of a clever data structure like a cover tree [BKL06] to store the centroids at each level. Then the nearest-neighbor matching of input patterns to the library (centroid) patterns would be very rapid, much faster than linearly comparing the input to each pattern in the list.

28.3.1.1 Conceptual Justification for Uniform DeSTIN

Generally speaking, one may say that: **if** the class of images that the system will see is invariant with respect to linear translations, **then** without loss of generality, we can assume that the library of patterns at each node on the same level is the same.

In reality this assumption isn't quite going to hold. For instance, for an eye attached to a person or humanoid robot, the top of the pixel grid will probably look at a person's hair more often than the bottom ... because the person stands right-side-up more often than they stand upside-down, and because they will often fixate the center of their view on a person's face, etc. For this reason, we can recognize our friend's face better if we're looking at them directly, with their face centered in our vision.

However, we suggest that this kind of peculiarity is not really essential to vision processing for general intelligence. There's no reason you can't have an intelligent vision system that recognizes a face just as well whether it's centered in the visual field or not. (In fact you could straightforwardly explicitly introduce this kind of bias within a translation-invariant DeSTIN, but it's not clear this is a useful direction.)

By and large, in almost all cases, it seems to us that in a DeSTIN system exposed to a wide variety of real-world inputs in complex situations, the library of patterns in the different nodes at the same level would turn out to be substantially the same. Even if they weren't exactly the same, they would be close to the same, embodying essentially the same regularities. But of course, this sameness would be obscured, because centroid 7 in a certain node X on level 4 might actually be the same as centroid 18 in some other node Y on level 4 ... and there would be no way to tell that centroid 7 in node X and centroid 18 and node Y were actually referring to the same pattern, without doing a lot of work.

28.3.1.2 Comments on Biological Realism

Translation-invariant DeSTIN deviates further from human brain structure than classic DeSTIN, but this is for good reason.

The brain has a lot of neurons, since adding new neurons was fairly easy and cheap for evolution; and tends to do things in a massively parallel manner,

with great redundancy. For the brain, it's not so problematically expensive to have the functional equivalent of a lot of DeSTIN nodes on the same level, all simultaneously using and learning libraries of patterns that are essentially identical to each other. Using current computer technology, on the other hand, this sort of strategy is rather inefficient.

In the brain, messaging between separated regions is expensive, whereas replicating function redundantly is cheap. In current computers, messaging between separated regions is fairly cheap (so long as those regions are stored on the same machine), whereas replicating function redundantly is expensive. Thus, even in cases where the same concept and abstract mathematical algorithm can be effectively applied in both the brain and a computer, the specifics needed for efficient implementation may be quite different.

28.3.2 Mapping States of Translation-Invariant DeSTIN into the Atomspace

Mapping classic DeSTIN's states into a symbolic pattern-manipulation engine like OpenCog is possible, but relatively cumbersome. Doing the same thing with Uniform DeSTIN is much more straightforward.

In Uniform DeSTIN, for example, Cluster 7 means the same thing in ANY node on level 4. So after a Uniform DeSTIN system has seen a fair number of images, you can be pretty sure its library of patterns is going to be relatively stable. Some clusters may come and go as learning progresses, but there's going to be a large and solid library of clusters at each level that persists, because all of its member clusters occur reasonably often across a variety of inputs.

Define a DeSTIN state-tree as a (quaternary) tree with one node for each DeSTIN node; and living at each node, a small list of (integer pattern_code, float weight) pairs. That is, at each node, the state-tree has a short-list of the patterns that closely match a given state at that node. The weights may be assumed between 0 and 1. The integer pattern codes have the same meaning for every node on the same level.

As you feed DeSTIN inputs, at each point in time it will have a certain state, representable as a state-tree. So, suppose you have a large database of DeSTIN state-trees, obtained by showing various inputs to DeSTIN over a long period of time. Then, you can do various kinds of pattern recognition on this database of state-trees.

More formally, define a state-subtree as a (quaternary) tree with a single integer at each node. Two state-subtrees may have various relationships with each other within a single state-tree – for instance they may be adjacent to each other, or one may appear atop or below the other, etc. In these terms, one interesting kind of pattern recognition to do is: Recognize frequent state-subtrees in the stored library of state-trees; and then recognize frequent rela-

tionships between these frequent state-subtrees. The latter relationships will form a kind of "image grammar," conceptually similar and formally related to those described in [ZM06]. Further, temporal patterns may be recognized in the same way as spatial ones, as part of the state-subtree grammar (e.g. state-subtree A often occurs right before state-subtree B; state-subtree C often occurs right before and right below state-subtree D; etc.).

The flow of activation from OpenCog back down to DeSTIN is also fairly straightforward in the context of translation-invariant DeSTIN. If relationships have been stored between concepts in OpenCogPrimes memory and grammatical patterns between state-subtrees, then whenever concept C becomes important in OpenCogPrimes memory, this can cause a top-down increase in the probability of matching inputs to DeSTIN node centroids, that would cause the DeSTIN state-tree to contain the grammatical patterns corresponding to concept C.

28.3.3 Scale-Invariant DeSTIN

The next step, moving beyond translation invariance, is to make DeSTIN's pattern recognition mostly (not wholly) scale invariant. We will describe a straightforward way to map centroids on one level of DeSTIN, into centroids on the other levels of DeSTIN. This means that when a centroid has been learned on one level, it can be experimentally ported to all the other levels, to see if it may be useful there too.

To make the explanation of this mapping clear, we reiterate some DeSTIN basics in slightly different language:

- A centroid on Level N is: a spatial arrangement (e.g. $k \times k$ square lattice) of beliefs of Level $N-1$. (More generally it is a spatiotemporal arrangement of such beliefs, but we will ignore this for the moment.)
- A belief on Level N is: a probability distribution over centroids on Level N. For heuristic purposes one can think about this as a mixture of Gaussians, though this won't always be the best model.
- Thus, a belief on Level N is: a probability distribution over spatial (or more generally, spatiotemporal) arrangements of beliefs on Level $N-1$

On Level 1, the role of centroids is played by simple $k \times k$ squares of pixels. Level 1 beliefs are probability distributions over these small pixel squares. Level 2 centroids are hence spatial arrangements of probability distributions over small pixel-squares; and Level 2 beliefs are probability distributions over spatial arrangements of probability distributions over small pixel-squares.

A small pixel-square S may be mapped into a single pixel P via a heuristic algorithm such as:

- if S has more black than white pixels, then P is black

- is S has more white than black pixels, then P is white
- if S has an equal number of white and black pixels, then use some heuristic. For instance if S is 4×4 you could look at the central 2×2 square and assign P to the color that occurs most often there. If that is also a tie, then you can just arbitrarily assign P to the color that occurs in the upper left corner of S.

A probability distribution over small pixel-squares may then be mapped into a probability distribution over pixel values (B or W). A probability distribution over the two values B and Wmay be approximatively mapped into a single pixel value – the one that occurs most often in the distribution, with a random choice made to break a tie. This tells us how to map Level 2 beliefs into spatial arrangements of pixels; and thus, it tells us how to map Level 2 beliefs into Level 1 beliefs.

But this tells us how to map Level N beliefs into Level $N-1$ beliefs, inductively. Remember, a Level N belief is a probability distribution (pdf for short) over spatial arrangements of beliefs on Level $N-1$. For example: A Level 3 belief if a pdf over arrangements of Level 2 beliefs. But since we can map Level 2 beliefs into Level 1 beliefs, this means we can map a Level 3 belief into a pdf over arrangements of Level 1 beliefs – which means we can map a Level 3 belief into a Level 2 belief. Etc.

Of course, this also tells us how to map Level N centroids into Level $N-1$ centroids. A Level N centroid is a pdf over arrangements of Level $N-1$ beliefs; a Level $N-1$ centroid is a pdf over arrangements of Level $N-2$ beliefs. But Level $N-1$ beliefs can be mapped into Level $N-2$ beliefs, so Level N centroids can be represented as pdfs over arrangements of Level N beliefs, and hence mapped into Level $N-1$ centroids.

In practice, one can implement this idea by moving from the bottom up. Given the mapping from Level 1 "centroids" to pixels, one can iterate through the Level 1 beliefs and identify which pixels they correspond to. Then one can iterate through the Level 2 beliefs and identify which Level 1 beliefs they correspond to. Etc. Each Level N belief can be explicitly linked to a corresponding level $N-1$ belief. Synchronously, as one moves up the hierarchy, Level N centroids can be explicitly linked to corresponding Level $N-1$ centroids.

Since there are in principle more possible Level Nbeliefs than Level $N-1$ beliefs, the mapping from level Nbeliefs to level $N-1$ beliefs is many-to-one. This is a reason not to simply maintain a single centroid pool across levels. However, when a new centroid C is added to the Level N pool, it can be mapped into a Level $N-1$ centroid to be added to the Level $N-1$ pool (if not there already). And, it can also be used to spawn a Level $N+1$ centroid, drawn randomly from the set of possible Level $N+1$ centroids that map into C.

Also, note that it is possible to maintain a single centroid *numbering system* across levels, so that a reference like "centroid # 175" has only one meaning

in an entire DeSTIN network, even though some of these centroid may only be meaningful above a certain level in the network.

28.3.4 Rotation Invariant DeSTIN

With a little more work, one can make DeSTIN rotation and shear invariant as well [1]. Considering rotation first:

- When comparing an input A to a Level N node with a Level N centroid B, consider various rotations of A, and see which rotation gives the closest match.
- When you match a centroid to an input observation-or-belief, record the rotation angle corresponding to the match.

The second of these points implies the tweaked definitions

- A centroid on Level N is: a spatial arrangement (e.g. $k \times k$ square lattice) of beliefs of Level $N-1$
- A belief on Level N is: a probability distribution over (angle, centroid) pairs on Level N.

From these it follows that a belief on Level N is: a probability distribution over (angle, spatial arrangement of beliefs) pairs on Level $N-1$

An additional complexity here is that two different (angle, centroid) pairs (on the same level) could be (exactly or approximately) equal to each other. This necessitates an additional step of "centroid simplification", in which ongoing checks are made to see if there are any two centroids C_1, C_2 on the same level so that: There exist angles A_1, A_2 so that (A_1, C_1) is very close to (A_2, C_2). In this case the two centroids may be merged into one.

To apply these same ideas to shear, one may simply replace "rotation angle" in the above by "(rotation angle, shear factor) pair."

28.3.5 Temporal Perception

Translation and scale invariant DeSTIN can be applied perfectly well if the inputs to DeSTIN, at level 1, are movies rather than static images. Then, in the simplest version, Level 1 consists of pixel cubes instead of pixel squares, etc. (the third dimension in the cube representing time). The scale invariance achieved by the methods described above would then be scale invariance in time as well as in space.

---

[1] The basic idea in this section, in the context of rotation, is due to Jade O'Neill (private communication)

In this context, one may enable rectangular shapes as well as cubes. That is, one can look at a Level N centroid consisting of m time-slices of a $k \times k$ arrangement of Level $N-1$ beliefs – without requiring that $m = k$ .... This would make the centroid learning algorithm a little more complex, because at each level one would want to consider centroids with various values of m, from $m = 1, ..., k$ (and potentially $m > k$ also).

28.4 Interpretation of DeSTIN's Activity

Uniform DeSTIN constitutes a substantial change in how DeSTIN does its business of recognizing patterns in the world – conceptually as well as technically. To explicate the meaning of these changes, we briefly present our favored interpretation of DeSTIN's dynamics.

The centroids in the DeSTIN library represent points in "spatial pattern space", i.e. they represent exemplary spatial patterns. DeSTIN's beliefs, as probability distributions over centroids, represent guesses as to which of the exemplary spatial patterns are the best models of what's currently being seen in a certain space-time region.

This matching between observations and centroids might seem to be a simple matter of "nearest neighbor matching"; but the subtle point is, it's not immediately obvious how to best measure the distance between observations and centroids. The optimal way of measuring distance is going to depend on context; that is to say, on the actual distribution of observations in the system's real environment over time.

DeSTIN's algorithm for calculating the belief at a node, based on the observation and centroids at that node **plus** the beliefs at other nearby nodes, is essentially a way of tweaking the distance measurement between observations and centroids, so that this measurement accounts for the context (the historical distribution of observations). There are many possible ways of doing this tweaking. Ideally one could use probability theory explicitly, but that's not always going to be computationally feasible, so heuristics may be valuable, and various versions of DeSTIN have contained various heuristics in this regard.

The various ways of "uniformizing" DeSTIN described above (i.e. making its pattern recognition activity approximately invariant with respect to affine transformations), don't really affect this story – they just improve the algorithm's ability to learn based on small amounts of data (and its rapidity at learning from data in general), by removing the need for the system to repeatedly re-learn transformed versions of the same patterns. So the uniformization just lets DeSTIN carry out its basic activity faster and using less data.

28.4.1 DeSTIN's Assumption of Hierarchical Decomposability

Roughly speaking, DeSTIN will work well to the extent that: The average distance between each part of an actually observed spatial pattern, and the closest centroid pattern, is not too large (note: the choice of distance measure in this statement is potentially subtle). That is: DeSTIN's set of centroids is supposed to provide a compact model of the probability distribution of spatial patterns appearing in the experience of the cognitive system of which DeSTIN is a part.

DeSTIN's effective functionality relies on the assumption that this probability distribution is hierarchically decomposable – i.e. that the distribution of spatial patterns appearing over a $k \times k$ region can be compactly expressed, to a reasonable degree of approximation, as a spatial combination of the distributions of spatial patterns appearing over $(k/4) \times (k/4)$ regions. This assumption of hierarchical decomposability greatly simplifies the search problem that DeSTIN faces, but also restricts DeSTIN's capability to deal with more general spatial patterns that are not easily hierarchically decomposable. However, the benefits of this approach seem to outweigh the costs, given that visual patterns in the environments humans naturally encounter do seem (intuitively at least) to have this hierarchical property.

28.4.2 Distance and Utility

Above we noted that choice of distance measure involved in the assessment of DeSTIN's effective functionality is subtle. Further above, we observed that the function of DeSTIN's belief assessment is basically to figure out the contextually best way to measure the distance between the observation and the centroids at a node. These comments were both getting at the same point.

But what is the right measure of distance between two spatial patterns? Ultimately, the right measure is: the probability that the two patterns A and B can be used in the same way. That is: the system wants to identify observation A with centroid B if it has useful action-patterns involving B, and it can substitute A for B in these patterns without loss.

This is difficult to calculate in general, though. A rough proxy, which it seems will often be acceptable, is to measure the distance between A and B in terms of both

- the basic (extensional) distance between the physical patterns they embody (e.g. pixel by pixel distance)
- the contextual (intensional) distance, i.e. the difference between the contexts in which they occur

Via enabling the belief in a node's parent to play a role in modulating a certain node's belief, DeSTIN's core algorithm enables contextual/intensional factors to play a role in distance assessment.

28.5 Benefits and Costs of Uniform DeSTIN

We now summarize the main benefits and costs of Uniform DeSTIN a little more systematically. The key point we have made here regarding Uniform DeSTIN and representational transparency may be summarized as follows:

- Define an "affine perceptual equivalence class" as a set of percepts that are equivalent to each other, or nearly so, under affine transformation. An example would be views of the same object from different perspectives or distances.
- Suppose one has embodied agent using DeSTIN for visual perception, whose perceptual stream tends to include a lot of reasonably large affine perceptual equivalence classes.
- Then, supposing the "mechanics" of DeSTIN can be transferred to the Uniform DeSTIN case without dramatic loss of performance, Uniform DeSTIN should be able to recognize patterns based on many fewer examples than classic DeSTIN

As soon as Uniform DeSTIN has learned to recognize one element of a given affine perceptual equivalence class, it can recognize all of them. Whereas, classic DeSTIN must learn each element of the equivalence class separately. So, roughly speaking, the number of cases required for unsupervised training of Uniform DeSTIN will be less than that for classic DeSTIN, by a ratio equal to the average size of the affine perceptual equivalence classes in the agent's perceptual stream.

Counterbalancing this, we have the performance cost of comparing the input to each node against a much larger set of centroids (in Uniform DeSTIN as opposed to classic DeSTIN). However, if a cover tree or other efficient data structure is used, this cost is not so onerous. The cost of nearest neighbor queries in a cover tree storing n items (in this case, n centroids) is $O(c^{12}logn)$, where the constant c represents the "intrinsic dimensionality" of the data; and in practice the cover tree search algorithm seems to perform quite well. So, the added time cost for online clustering in Uniform DeSTIN as opposed to DeSTIN, is a factor on the order of the log of the number of nodes in the DeSTIN tree. We believe this moderate added time cost is well worth paying, to gain a significant decrease in the number of training examples required for unsupervised learning.

Beyond increases in computational cost, there is also the risk that the online clustering may just not work as well when one has so many clusters in each node. This is the sort of problem that can really only be identified, and

dealt with, during extensive practice – since the performance of any clustering algorithm is largely determined by the specific distribution of the data it's dealing with. It may be necessary to improve DeSTIN's online clustering in some way to make Uniform DeSTIN work optimally, e.g. improving its ability to form clusters with markedly non-spherical shapes. This ties in to a point raised in chapter 29 – the possibility of supplementing traditional clusters with predicates learned by OpenCog, which may live inside DeSTIN nodes alongside centroids. Each such predicate in effect defines a (generally nonconvex) "cluster".

28.6 Imprecise Probability as a Strategy for Linking CogPrime and DeSTIN

One key aspect of vision processing is the ability to preferentially focus attention on certain positions within a perceived visual scene. In this section we describe a novel strategy for enabling this in a hybrid CogPrime /DeSTIN system, via use of imprecise probabilities. In fact the basic idea suggested here applies to any probabilistic sensory system, whether deep-learning-based or not, and whether oriented toward vision or some other sensory modality. However, for sake of concreteness, we will focus here on the case of DeSTIN/CogPrime integration.

28.6.1 Visual Attention Focusing

Since visual input streams contain vast amounts of data, it's beneficial for a vision system to be able to focus its attention specifically on the most important parts of its input. Sometimes knowledge of what's important will come from cognition and long-term memory, but sometimes it may come from mathematical heuristics applied to the visual data itself.

In the human visual system the latter kind of "low level attention focusing" is achieved largely in the context of the eye changing its focus frequently, looking preferentially at certain positions in the scene [Cha09]. This works because the center of the eye corresponds to a greater density of neurons than the periphery.

So for example, consider a computer vision algorithm like SIFT (Scale-Invariant Feature Extraction) [Low99], which (as shown in Figure 28.1) mathematically isolates certain points in a visual scene as ÒkeypointsÓ which are particularly important for identifying what the scene depicts (e.g. these may be corners, or easily identifiable curves in edges). The human eye, when looking at a scene, would probably spend a greater percentage of its time focusing on the SIFT keypoints than on random points in the image.

The human visual system's strategy for low-level attention focusing is obviously workable (at least in contexts similar to those in which the human eye evolved), but itÕs also somewhat complex, requiring the use of subtle temporal processing to interpret even static scenes. We suggest here that there may be a simpler way to achieve the same thing, in the context of vision systems that are substantially probabilistic in nature, via using imprecise probabilities. The crux of the idea is to represent the most important data, e.g. keypoints, using imprecise probability values with greater confidence.

Similarly, cognition-guided visual attention-focusing occurs when a mind's broader knowledge of the world tells it that certain parts of the visual input may be more interesting to study than others. For example, in a picture of a person walking down a dark street, the contours of the person may not be tremendously striking visually (according to SIFT or similar approaches); but even so, if the system as a whole knows that it's looking at a person, it may decide to focus extra visual attention on anything person-like. This sort of cognition guided visual attention focusing, we suggest, may be achieved similarly to visual attention focusing guided on lower-level cues – by increasing the confidence of the imprecise probabilities associated with those aspects of the input that are judged more cognitively significant.

28.6.2 Using Imprecise Probabilities to Guide Visual Attention Focusing

Suppose one has a vision system that internally constructs probabilistic values corresponding to small local regions in visual input (these could be pixels or voxels, or something a little larger), and then (perhaps via a complex process) assigns probabilities to different interpretations of the input based on combinations of these input-level probabilities. For this sort of vision system, one may be able to achieve focusing of attention via appropriately replacing the probabilities with imprecise probabilities. Such an approach may be especially interesting in hierarchical vision systems, that also involve the calculation of probabilities corresponding to larger regions of the visual input. Examples of the latter include deep learning based vision systems like HTM or DeSTIN, which construct nested hierarchies corresponding to larger and larger regions of the input space, and calculate probabilities associated with each of the regions on each level, based in part on the probabilities associated with other related regions.

In this context, we now state the basic suggestion of the section:

1. Assign higher confidence to the low-level probabilities that the vision system creates corresponding to the local visual regions that one wants to focus attention on (based on cues from visual preprocessing or cognitive guidance)

2. Carry out the vision system's processing using imprecise probabilities rather than single-number probabilities
3. Wherever the vision system makes a decision based on Òthe most probable choiceÓ from a number of possibilities, change the system to make a decision based on Òthe choice maximizing the product (expectation * confidence)Ó.

28.6.3 Sketch of Application to DeSTIN

Internally to DeSTIN, probabilities are assigned to clusters associated with local regions of the visual input. If a system such as SIFT is run as a preprocessor to DeSTIN, then those small regions corresponding to SIFT keypoints may be assumed semantically meaningful, and internal DeSTIN probabilities associated with them can be given a high confidence. A similar strategy may be taken if a cognitive system such as OpenCog is run together with DeSTIN, feeding DeSTIN information on which portions of a partially-processed image appear most cognitively relevant. The probabilistic calculations inside DeSTIN can be replaced with corresponding calculations involving imprecise probabilities. And critically, there is a step in DeSTIN where, among a set of beliefs about the state in each region of an image (on each of a set of hierarchical levels), the one with the highest probability is selected. In accordance with the above recipe, this step should be modified to select the belief with the highest probability*confidence.

28.6.3.1 Conceptual Justification

What is the conceptual justification for this approach?

One justification is obtained by assuming that each percept has a certain probability of being erroneous, and those percepts that appear to more closely embody the semantic meaning of the visual scene are less likely to be erroneous. This follows conceptually from the assumption that the perceived world tends to be patterned and structured, so that being part of a statistically significant pattern is (perhaps weak) evidence of being real rather than artifactual. Under this assumption, the proposed approach will maximize the accuracy of the systemÕs judgments.

A related justification is obtained by observing that this algorithmic approach follows from the consideration of the perceived world as mutable. Consider a vision system that has the capability to modify even the low-level percepts that it intakes Ð i.e. to use what it thinks and knows, to modify what it sees. The human brain certainly has this potential [Cha09]. In this case, it will make sense for the system to place some constraints regarding which of its percepts it is more likely to modify. Confidence values semanti-

cally embody this Ð a higher confidence being sensibly assigned to percepts that the system considers should be less likely to be modified based on feedback from its higher (more cognitive) processing levels. In that case, a higher confidence should be given to those percepts that seem to more closely embody the semantic meaning of the visual scene Ð which is exactly what we're suggesting here.

28.6.3.2 Enabling Visual Attention Focusing in DeSTIN via Imprecise Probabilities

We now refer back to the mathematical formulation of DeSTIN summarized in Section 4.3.1 of Chapter 4 above, in the context of which the application of imprecise probability based attention focusing to DeSTIN is almost immediate.

The probabilities $P(o|s)$ may be assigned greater or lesser confidence depending on the assessed semantic criticality of the observation o in question. So for instance, if one is using SIFT as a preprocessor to DeSTIN, then one may assign probabilities $P(o|s)$ higher confidence if they correspond to observations o of SIFT keypoints, than if they do not.

These confidence levels may then be propagated throughout DeSTIN's probabilistic mathematics. For instance, if one were using Walley's interval probabilities, then one could carry out the probabilistic equations using interval arithmetic.

Finally, one wishes to replace Equation 4.3.1.2 in Chapter 4 with

$$c = \arg\max_s \left((b_p(s)).\text{strength} * (b_p(s)).\text{confidence} \right), \tag{28.1}$$

or some similar variant. The effect of this is that hypotheses based on high-confidence observations are more likely to be chosen, which of course has a large impact on the dynamics of the DeSTIN network.

Fig. 28.1 The SIFT algorithm finds keypoints in an image, i.e. localized features that are particularly useful for identifying the objects in an image. The top row shows images that are matched against the image in the middle row. The bottom-row image shows some of the keypoints used to perform the matching (i.e. these keypoints demonstrate the same features in the top-row images and their transformed middle-row counterparts). SIFT keypoints are identified via a staged filtering approach. The first stage identifies key locations in scale space by looking for locations that are maxima or minima of a difference-of-Gaussian function. Each point is used to generate a feature vector that describes the local image region sampled relative to its scale-space co- ordinate frame. The features achieve partial invariance to local variations, such as affine or 3D projections, by blurring image gradient locations.

Chapter 29
Bridging the Symbolic/Subsymbolic Gap

29.1 Introduction

While it's widely accepted that human beings carry out both *symbolic* and *subsymbolic* processing, as integral parts of their general intelligence, the precise definition of "symbolic" versus "subsymbolic" is a subtle issue, which different AI researchers will approach in different ways depending on their differing overall perspectives on AI. Nevertheless, the intuitive meaning of the concepts is commonly understood:

- **"subsymbolic"** refers to things like pattern recognition in high-dimensional quantitative sensory data, and real-time coordination of multiple actuators taking multidimensional control signals
- **"symbolic"** refers to things like natural language grammar and (certain or uncertain) logical reasoning, that are naturally modeled in terms of manipulation of symbolic tokens in terms of particular (perhaps experientially learned) rules

Views on the relationship between these two aspects of intelligence in human and artificial cognition are quite diverse, including perspectives such as

1. Symbolic representation and reasoning are the core of human-level intelligence; subsymbolic aspects of intelligence are of secondary importance and can be thought of as pre or post processors to symbolic representation and reasoning
2. Subsymbolic representation and learning are the core of human intelligence; symbolic aspects of intelligence
    a. emerge from the subsymbolic aspects as needed; or,
    b. arise via a relatively simple, thin layer on top of subsymbolic intelligence, that merely applies subsymbolic intelligence in a slightly different way

3. Symbolic and subsymbolic aspects of intelligence are best considered as different subsystems, which
    a. have a significant degree of independent operation, but also need to coordinate closely together; or,
    b. operate largely separately and can be mostly considered as discrete modules

In evolutionary terms, it is clear that subsymbolic intelligence came first, and that most of the human brain is concerned with the subsymbolic intelligence that humans share with other animals. However, this observation doesn't have clear implications regarding the relationship between symbolic and subsymbolic intelligence in the context of everyday cognition.

In the history of the AI field, the symbolic/subsymbolic distinction was sometimes aligned with the dichotomy between logic-based and rule-based AI systems (on the symbolic side) and neural networks (on the subsymbolic side) [PJ88b]. However, this dichotomy has become much blurrier in the last couple decades, with developments such as neural network models of language parsing [GH11] and logical reasoning [LBH10], and symbolic approaches to perception and action [SR04]. Integrative approaches have also become more common, with one of the major traditional symbolic AI systems, ACT-R, spawning a neural network version [LA93] with parallel structures and dynamics to the traditional explicitly symbolic version and a hybridization with a computational neuroscience model [JL08]; and another one, SOAR, incorporating perception processing components as separate modules [Lai12]. The field of "neural-symbolic computing" has emerged, covering the emergence of symbolic rules from neural networks, and the hybridization of neural networks with explicitly symbolic systems [HH07].

Our goal here is not to explore the numerous deep issues involved with the symbolic/subsymbolic dichotomy, but rather to describe the details of a particular approach to symbolic/subsymbolic integration, inspired by Perspective 3a in the above list: the consideration of symbolic and subsymbolic aspects of intelligence as different subsystems, which have a significant degree of independent operation, but also need to coordinate closely together. We believe this kind of integration can serve a key role in the quest to create human-level general intelligence. The approach presented here is at the beginning rather than end of its practical implementation; what we are describing here is the initial design intention of a project in progress, which is sure to be revised in some respects as implementation and testing proceed. We will focus mainly on the tight integration of a subsymbolic system enabling gray-scale vision processing into a cognitive architecture with significant symbolic aspects, and will then briefly explain how the same ideas can be used for color vision, and multi-sensory and perception-action integration.

The approach presented here begins with two separate AI systems, both currently implemented in open-source software:

- **OpenCog**, an integrative architecture for AGI [Goe10d] [GPW+11], which is centered on a "weighted, labeled hypergraph" knowledge representation called the Atomspace, and features a number of different, sophisticated cognitive algorithms acting on the Atomspace. Some of these cognitive algorithms are heavily symbolic in focus (e.g. a probabilistic logic engine); others are more subsymbolic in nature (e.g. a neural net like system for allocating attention and assigning credit). However, OpenCog in its current form cannot deal with high-dimensional perceptual input, nor with detailed real-time control of complex actuators. OpenCog is now being used to control intelligent characters in an experimental virtual world, where the perceptual inputs are the 3D coordinate locations of objects or small blocks; and the actions are movement commands like "step forward", "turn head to the right."
- **DeSTIN** [ARK09a],[ARC09], a deep learning system consisting of a hierarchy of processing nodes, in which the nodes on higher levels correspond to larger regions of space-time, and each node carries out prediction regarding events in the space-time region to which it corresponds. Feedback and feedforward dynamics between nodes combine with the predictive activity within nodes, to create a complex nonlinear dynamical system whose state self-organizes to reflect the state of the world being perceived. The core concepts of DeSTIN are similar to those of Jeff Hawkins' Numenta system [HB06] [GH09], Dileep George's work (http://vicariousinc.com) and work by Mohamad Tarifi [TSH11], Bundzel and Hashimoto [BH10], and others. However, the specifics of DeSTIN's dynamics have been designed in what we consider a particularly powerful way, and the system has shown good results on small-scale test problems [KAR10]. So far DeSTIN has been utilized only for vision processing, but a similar proprietary system has been used for auditory data as well; and DeSTIN was designed to work together with an accompanying action hierarchy.

These two systems were not originally designed to work together, but we will describe a method for achieving their tight integration via

1. Modifying DeSTIN in several ways, so that
   a. the patterns in its states over time will have more easily recognizable regularities
   b. its nodes are able to scan their inputs not only for simple statistical patterns (DeSTIN "centroids"), but also for patterns recognized by routines supplied to it by an external source (e.g. another AI system such as OpenCog)
2. Utilizing one of OpenCogPrimes cognitive processes (the "Fishgram" frequent subhypergraph mining algorithm) to recognize patterns in sets of DeSTIN states, and then recording these patterns in OpenCogPrimes Atomspace knowledge store

3. Utilizing OpenCogPrimes other cognitive processes to abstract concepts and draw conclusions from the patterns recognized in DeSTIN states by Fishgram
4. Exporting the concepts and conclusions thus formed to DeSTIN, so that its nodes can explicitly scan for their presence in their inputs, thus allowing the results of symbolic cognition to explicitly guide subsymbolic perception
5. Creating an action hierarchy corresponding closely to DeSTIN's perceptual hierarchy, and also corresponding to the actuators of a particular robot. This allows action learning to be done via an optimization approach ([LKP+05], [YKL+04]), where the optimization algorithm uses DeSTIN states corresponding to perceived actuator states as part of its inputs.

The ideas presented here are compatible with those described in [Goe11a], but different in emphasis. That paper described a strategy for integrating OpenCog and DeSTIN via creating an intermediate "semantic CSDLN" hierarchy to translate between OpenCog and DeSTIN, in both directions. In the approach suggested here, this semantic CSDLN hierarchy exists conceptually but not as a separate software object: it exists as the combination of

- OpenCog predicates exported to DeSTIN and used alongside DeSTIN centroids, inside DeSTIN nodes
- OpenCog predicates living in the OpenCog knowledge repository (AtomSpace), and interconnected in a hierarchical way using OpenCog nodes and links (thus reflecting DeSTIN's hierarchical structure within the AtomSpace).

This hierarchical network of predicates, spanning the two software systems, plays the role of a semantic CSDLN as described in [Goe11a].

29.2 Simplified OpenCog Workflow

The dynamics inside an OpenCog system may be highly complex, defying simple flowcharting, but from the point of view of OpenCog-DeSTIN integration, one important pattern of information flow through the system is as follows:

1. Perceptions come into the Atomspace. In the current OpenCog system, these are provided via a proxy to the game engine where the OpenCog controlled character interacts. In an OpenCog-DeSTIN hybrid, these will be provided via DeSTIN.
2. Hebbian learning builds HebbianLinks between perceptual Atoms representing percepts that have frequently co-occurred

3. PLN inference, concept blending and other methods act on these perceptual Atoms and their HebbianLinks, forming links between them and linking them to other Atoms stored in the Atomspace reflecting prior experience and generalizations therefrom
4. Attention allocation gives higher short and long term importance values to those Atoms that appear likely to be useful based on the links they have obtained
5. Based on the system's current goals and subgoals (the latter learned from the top-level goals using PLN), and the goal-related links in the Atomspace, the OpenPsi mechanism triggers the PLN-based planner, which chooses a series of high-level actions that are judged likely to help the system achieve its goals in the current context
6. The chosen high-level actions are transformed into series of lower-level, directly executable actions. In the current OpenCog system, this is done by a set of hand-coded rules based on the specific mechanics of the game engine where the OpenCog controlled character interacts. In an OpenCog-DeSTIN hybrid, the lower-level action sequence will be chosen by an optimization method acting based on the motor control and perceptual hierarchies.

This pattern of information flow omits numerous aspects of OpenCog cognitive dynamics, but gives the key parts of the picture in terms of the interaction of OpenCog cognition with perception and action. Most of the other aspects of the dynamics have to do with the interaction of multiple cognitive processes acting on the Atomspace, and the interaction between the Atomspace and several associated specialized memory stores, dealing with procedural, episodic, temporal and spatial aspects of knowledge. From the present point of view, these additional aspects may be viewed as part of Step 3 above, wrapped up in the phrase "É and other methods act on these perceptual Atoms." However, it's worth noting that in order to act appropriately on perceptual Atoms, a lot of background cognition regarding more abstract conceptual Atoms (often generalized from previous perceptual Atoms) may be drawn on. This background inference incorporates both symbolic and subsymbolic aspects, but goes beyond the scope of the present discussion, as its particulars do not impinge on the particulars of DeSTIN-OpenCog integration.

OpenCog also possesses a specialized facility for natural language comprehension and generation [LGE10] [Goe10c], which may be viewed as a parallel perception/action pathway, bypassing traditional human-like sense perception and dealing with text directly. Integrating OpenCogPrimes current linguistics processes with DeSTIN-based auditory and visual processing is a deep and important topic, but one we will bypass here, for sake of brevity and because it's not our current research priority.

29.3 Integrating DeSTIN and OpenCog

The integration of DeSTIN and OpenCog involves two key aspects:

- recognition of patterns in sets of DeSTIN states, and exportation of these patterns into the OpenCog Atomspace
- use of OpenCog-created concepts within DeSTIN nodes, alongside statistically-derived "centroids"

From here on, unless specified otherwise, when we mention "DeSTIN" we will refer to "Uniform DeSTIN" as defined in the companion paper [Goeon], an extension of "classic DeSTIN" as defined in [ARK09a].

29.3.1 Mining Patterns from DeSTIN States

The first step toward using OpenCog tools to mine patterns from sets of DeSTIN states, is to represent these states in Atom form in an appropriate way. A simple but workable approach, restricting attention for the moment to purely spatial patterns, is to use the six predicates:

- $hasCentroid(node\ N, int\ k)$
- $hasParentCentroid(node\ N, int\ k)$
- $hasNorthNeighborCentroid(node\ N, int\ k)$
- $hasSouthNeighborCentroid(node\ N, int\ k)$
- $hasEastNeighborCentroid(node\ N, int\ k)$
- $hasWestNeighborCentroid(node\ N, int\ k)$

For instance

$$hasNorthNeighborCentroid(N, 3)$$

means that N's north neighbor has centroid #3

One may consider also the predicates

- $hasParent(node\ N, Node\ M)$
- $hasNorthNeighbor(node\ N, Node\ M)$
- $hasSouthNeighbor(node\ N, Node\ M)$
- $hasEastNeighbor(node\ N, Node\ M)$
- $hasWestNeighbor(node\ N, Node\ M)$

Now suppose we have a stored set of DeSTIN states, saved from the application of DeSTIN to multiple different inputs. What we want to find are predicates P that are *conjunctions* of instances of the above 10 predicates, which occur frequently in the stored set of DeSTIN states. A simple example of such a predicate would be the conjunction of

- $hasNorthNeighbor(\$N, \$M)$
- $hasParentCentroid(\$N, 5)$
- $hasParentCentroid(\$M, 5)$
- $hasNorthNeighborCentroid(\$N, 6)$
- $hasWestNeighborCentroid(\$M, 4)$

This predicate could be evaluated at any pair of nodes $(\$N, \$M)$ on the same DeSTIN level. If it is true for atypically many of these pairs, then it's a "frequent pattern", and should be detected and stored.

OpenCogPrimes pattern mining component, Fishgram, exists precisely for the purpose of mining this sort of conjunction from sets of relationships that are stored in the Atomspace. It may be applied to this problem as follows:

- Translate each DeSTIN state into a set of relationships drawn from: hasNorthNeighbor, hasSouthNeighbor, hasEastNeighbor, hasWestNeighbor, hasCentroid, hasParent
- Import these relationships, describing each DeSTIN state, into the OpenCog Atomspace
- Run pattern mining on this AtomSpace.

29.3.2 Probabilistic Inference on Mined Hypergraphs

Patterns mined from DeSTIN states can then be reasoned on by OpenCogPrimes PLN inference engine, allowing analogy and generalization.

Suppose centroids 5 and 617 are estimated to be similar – either via DeSTIN's built-in similarity metric, or, more interestingly via OpenCog inference on the Atom representations of these centroids. As an example of the latter, consider: 5 could represent a person's nose and 617 could represent a rabbit's nose. In this case, DeSTIN might not judge the two centroids particularly similar on a purely visual level, but, OpenCog may know that the images corresponding to both of these centroids are are called "noses" (e.g. perhaps via noticing people indicate these images in association with the word "nose"), and may thus infer (using a simple chain of PLN inferences) that these centroids seem probabilistically similar.

If 5 and 617 are estimated to be similar, then a predicate like

```
ANDLink
   EvaluationLink
      hasNorthNeighbor
      ListLink $N  $M
   EvaluationLink
      hasParentCentroid
      ListLink $N  5
   EvaluationLink
```

```
        hasParentCentroid
        ListLink $M  5
     EvaluationLink
        hasNorthNeighborCentroid
        ListLink $N 6
     EvaluationLink
        hasWestNeighborCentroid
        ListLink $M 4
```

mined from DeSTIN states, could be extended via PLN analogical reasoning to

```
  ANDLink
     EvaluationLink
        hasNorthNeighbor
        ListLink $N $M
     EvaluationLink
        hasParentCentroid
        ListLink $N  617
     EvaluationLink
        hasParentCentroid
        ListLink $M  617
     EvaluationLink
        hasNorthNeighborCentroid
        ListLink $N 6
     EvaluationLink
        hasWestNeighborCentroid
        ListLink $M 4
```

29.3.3 Insertion of OpenCog-Learned Predicates into DeSTIN's Pattern Library

Suppose one has used Fishgram, as described above, to recognize predicates embodying frequent or surprising patterns in a set of DeSTIN states or state-sequences. The next natural step is to add these frequent or surprising patterns to DeSTIN's pattern library, so that the pattern library contains not only classic DeSTIN centroids, but also these corresponding "image grammar" style patterns. Then, when a new input comes into a DeSTIN node, in addition to being compared to the centroids at the node, it can be fed as input to the predicates associated with the node.

What is the advantage of this approach, compared to DeSTIN without these predicates? The capability for more compact representation of a variety of spatial patterns. In many cases, a spatial pattern that would require a large number of DeSTIN centroids to represent, can be represented by a

single, fairly compact predicate. It is an open question whether these sorts of predicates are really critical for human-like vision processing. However, our intuition is that they do have a role in human as well s machine vision. In essence, DeSTIN is based on a fancy version of nearest-neighbor search, applied in a clever way on multiple levels of a hierarchy, using context-savvy probabilities to bias the matching. But we suspect there are many visual patterns that are more compactly and intuitively represented using a more flexible language, such as OpenCog predicates formed by combining elementary predicates involving appropriate spatial and temporal relations.

For example, consider the archetypal spatial pattern of a face as: either two eyes that are next to each other, or sunglasses, above a nose, which is in turn above a mouth. (This is an oversimplified toy example, but we're positing it for illustration only. The same point applies to more complex and realistic patterns.) One could represent this in OpenCogPrimes Atom language as something like:

```
AND
   InheritanceLink N B_nose
   InheritanceLink M B_mouth
   EvaluationLink
       above
       ListLink E N
   EvaluationLink
      above
      ListLink N M
   OR
     AND
        MemberLink E1  E
        MemberLink E2  E
        EvaluationLink
           next_to
           ListLink E1 E2
        InheritanceLink E1 B_eye
     AND
        InheritanceLink E  B_sunglasses
```

where e.g. B_eye is a DeSTIN belief that corresponds roughly to recognition of the spatial pattern of a human eye. To represent this using ordinary DeSTIN centroids, one couldn't represent the OR explicitly; instead one would need to split it into two different sets of centroids, corresponding to the eye case and the sunglasses caseÉ unless the DeSTIN pattern library contained a belief corresponding to "eyes or sunglasses." But the question then becomes: how would classic DeSTIN actually learn a belief like this? In the suggested architecture, pattern mining on the database of DeSTIN states is proposed as an algorithm for learning such beliefs.

This sort of predicate-enhanced DeSTIN will have advantages over the traditional version, only if the actual distribution of images observed by the system contains many (reasonably high probability) images modeled accurately by predicates involving disjunctions and/or negations as well as conjunctions. If the system's perceived world is simpler than this, then good old DeSTIN will work just as well, and the OpenCog-learned predicates are a needless complication.

Without these sorts of predicates, how might DeSTIN be extended to include beliefs like "eyes or sunglasses"? One way would be to couple DeSTIN with a reinforcement learning subsystem, that reinforced the creation of beliefs that were useful for the system as a whole. If reasoning in terms of faces (independent of whether they have eyes or sunglasses) got the system reward, presumably it could learn to form the concept "eyes or sunglasses." We believe this would also be a workable approach, but that given the strengths and weaknesses of contemporary computer hardware, the proposed DeSTIN-OpenCog approach will prove considerably simpler and more effective.

29.4 Multisensory Integration, and Perception-Action Integration

In Part I we have briefly indicated how DeSTIN could be extended beyond vision to handle other senses such as audition and touch. If one had multiple perception hierarchies corresponding to multiple senses, the easiest way to integrate them within an OpenCog context would be to use OpenCog as the communication nexus – representing DeSTIN centroids in the various modality-specific hierarchies as OpenCog Atoms (PerceptualCentroidNodes), and building HebbianLinks in OpenCogPrimes Atomspace between these PerceptualCentroidNodes as appropriate based on their association. So for instance the sound of a person's footsteps would correspond to a certain belief (probability distribution over centroids) in the auditory DeSTIN network, and the sight of a person's feet stepping would correspond to a certain belief (probability distribution over centroids) in the visual DeSTIN network; and the OpenCog Atomspace would contain links between the sets of centroids assigned high weights between these two belief distributions. Importance spreading between these various PerceptualCentroidNodes would cause a dynamic wherein seeing feet stepping would bias the system to think it was hearing footsteps, and hearing footsteps would bias it to think it was seeing feet stepping.

And, suppose there are similarities between the belief distributions for the visual appearance of dogs, and the visual appearance of cats. Via the intermediary of the Atomspace, this would bias the auditory and haptic DeSTIN hierarchies to assume a similarity between the auditory and haptic charac-

teristics of dogs, and the analogous characteristics of cats. Because: PLN analogical reasoning would extrapolate from, e.g.

- HebbianLinks joining cat-related visual PerceptualCentroidNodes and dog-related visual PerceptualCentroidNodes
- HebbianLinks joining cat-related visual PerceptualCentroidNodes to cat-related haptic PerceptualCentroidNodes; and others joining dog-related visual PerceptualCentroidNodes to dog-related haptic PerceptualCentroidNodes

to yield HebbianLinks joining cat-related haptic PerceptualCentroidNodes and dog-related haptic PerceptualCentroidNodes. This sort of reasoning would then cause the system DeSTIN to, for example, upon touching a cat, vaguely expect to maybe hear dog-like things. This sort of simple analogical reasoning will be right sometimes and wrong sometimes – a cat walking sounds a fair bit like a dog walking, and cat and dog growls sound fairly similar, but a cat meowing doesn't sound that much like a dog barking. More refined inferences of the same basic sort may be used to get the details right as the system explores and understands the world more accurately.

29.4.1 Perception-Action Integration

While experimentation with DeSTIN has so far been restricted to perception processing, the system was designed from the beginning with robotics applications in mind, involving integration of perception with action and reinforcement learning. As OpenCog already handles reinforcement learning on a high level (via OpenPsi), our approach to robot control using DeSTIN and OpenCog involves creating a control hierarchy parallel to DeSTIN's perceptual hierarchy, and doing motor learning using optimization algorithms guided by reinforcement signals delivered from OpenPsi and incorporating DeSTIN perceptual states as part of their input information.

Our initial research goal, where action is concerned, is not to equal the best purely control-theoretic algorithms at fine-grained control of robots carrying out specialized tasks, but rather to achieve basic perception / control / cognition integration in the rough manner of a young human child. A two year old child is not particularly well coordinated, but is capable of coordinating actions involving multiple body parts using an integration of perception and action with unconscious and deliberative reasoning. Current robots, in some cases, can carry out specialized actions with great accuracy, but they lack this sort of integration, and thus generally have difficulty effectively carrying out actions in unforeseen environments and circumstances.

We will create an action hierarchy with nodes corresponding to different parts of the robot body, where e.g. the node corresponding to an arm would have child nodes corresponding to a shoulder, elbow, wrist and hand; and the

node corresponding to a hand would have child nodes corresponding to the fingers of the hand; etc. Physical self-perception is then achieved by creating a DeSTIN "action-perception" hierarchy with nodes corresponding to the states of body components. In the simplest case this means the lowest-level nodes will correspond to individual servomotors, and their inputs will be numerical vectors characterizing servomotor states. If one is dealing with a robot endowed with haptic technology, e.g. Syntouch [FL12] fingertips, then numerical vectors characterizing haptic inputs may be used alongside these.

The configuration space of an action-perception node, corresponding to the degrees of freedom of the servomotors of the body part the node represents, may be approximated by a set of "centroid" vectors. When an action is learned by the optimization method used for this purpose, this involves movements of the servomotors corresponding to many different nodes, and thus creates a series of "configuration vectors" in each node. These configuration vector series may be subjected to online clustering, similar to percepts in a DeSTIN perceptual hierarchy. The result is a library of "codewords", corresponding to discrete trajectories of movement, associated with each node. The libraries may be shared by identical body parts (e.g. shared among legs, shared among fingers), but will be distinct otherwise. Each coordinated whole-body action thus results in a series of (node, centroid) pairs, which may be mined for patterns, similarly to the perception case.

The set of predicates needed to characterize states in this action-perception hierarchy is simpler than the one described for visual perception above; here one requires only

- $hasCentroid(node\ N, int\ k)$
- $hasParentCentroid(node\ N, int\ k)$
- $hasParent(node\ N, Node\ M)$
- $hasSibling(node\ N, Node\ M)$

and most of the patterns will involve specific nodes rather than node variables. The different nodes in a DeSTIN vision hierarchy are more interchangeable (in terms of their involvement in various patterns) than, say, a leg and a finger.

In a pure DeSTIN implementation, the visual and action-perception hierarchies would be directly linked. In the context of OpenCog integration, it is simplest to link the two via OpenCog, in a sense using cognition as a bridge between action and perception. It is unclear whether this strategy will be sufficient in the long run, but we believe it will be more than adequate for experimentation with robotic perceptual-motor coordination in a variety of everyday tasks. OpenCogPrimes Hebbian learning process can be used to find common associations between action-perception states and visual-perception states, via mining a data store containing time-stamped state records from both hierarchies.

Importance spreading along the HebbianLinks learned in this way can then be used to bias the weights in the belief states of the nodes in both hierarchies.

So, for example, the action-perception patterns related to clenching the fist, would be Hebbianly correlated with the visual-perception patterns related to seeing a clenched fist. When a clenched fist was perceived via servomotor data, importance spreading would increase the weighting of visual patterns corresponding to clenched fists, within the visual hierarchy. When a clenched fist was perceived via visual data, importance spreading would increase the weighting of servomotor data patterns corresponding to clenched fists, within the action-perception hierarchy.

29.4.2 Thought-Experiment: Eye-Hand Coordination

For example, how would DeSTIN-OpenCog integration as described here carry out a simple task of eye-hand coordination? Of course the details of such a feat, as actually achieved, would be too intricate to describe in a brief space, but it still is meaningful to describe the basic ideas. Consider the case of a robot picking up a block, in plain sight immediately in front of the robot, via pinching it between two fingers and then lifting it. In this case,

- The visual scene, including the block, is perceived by DeSTIN; and appropriate patterns in various DeSTIN nodes are formed
- Predicates corresponding to the distribution of patterns among DeSTIN nodes are activated and exported to the OpenCog Atomspace
- Recognition that a block is present is carried out, either by
  - PLN inference within OpenCog, drawing the conclusion that a block is present from the exported predicates, using ImplicationLinks comprising a working definition of a "block"
  - A predicate comprising the definition of "block", previously imported into DeSTIN from OpenCog and utilized within DeSTIN nodes as a basic pattern to be scanned for. This option would obtain only if the system had perceived many blocks in the past, justifying the automation of block recognition within the perceptual hierarchy.
- OpenCog, motivated by one of its higher-level goals, chooses "picking up the block" as subgoal. So it allocates effort to finding a procedure whose execution, in the current context, has a reasonable likelihood of achieving the goal of picking up the block. For instance, the goal could be curiosity (which might make the robot want to see what lies under the block), or the desire to please the agent's human teacher (in case the human teacher likes presents, and will reward the robot for giving it a block as a present), etc.
- OpenCog, based on its experience, uses PLN to reason that "grabbing the block" is a subgoal of "picking up the block"
- OpenCog utilizes a set of predicates corresponding to the desired state of "grabbing the the block" as a target for an optimization algorithm,

designed to figure out a series of servomotor actions that will move the robot's body from the current state to the target state. This is a relatively straightforward control theory problem.

- Once the chosen series of servomotor actions has been executed, the robot has its fingers poised around the block, ready to pick it up. At this point, the action-perception hierarchy perceives what is happening in the fingers. If the block is really being grabbed properly, then the fingers are reporting some force, due to the feeling of grabbing the block (haptic input is another possibility and would be treated similarly, but we will leave that aside for now). Importance spreads from these action-perception patterns into the Atomspace, and back down into the visual perception hierarchy, stimulating concepts and percepts related to "something is being grabbed by the fingers."
- If the fingers aren't receiving enough force, because the agent is actually only poking the block with one finger and grabbing the air with another finder, then the "something is being grabbed by the fingers" stimulation doesn't happen, and the agent is less sure it's actually grabbing anything. In that case it may withdraw its hand a bit, so that it can more easily assess its hand's state visually, and try the optimization-based movement planning again.
- Once the robot estimates the goal of grabbing the block has been successfully achieved, it proceeds to the next sub-subgoal, and asks the action-sequence optimizer to find a sequence of movements that will likely cause the predicates corresponding to "hold the block up" to obtain. It then executes this movement series and picks the block up in the air.

This simple example is a far cry from the perceptual-motor coordination involved in doing embroidery, juggling or serving a tennis ball. But we believe it illustrates, in a simple way, the same basic cognitive structures and dynamics used in these more complex instances.

29.5 Conclusion

We have described, at a high level, a novel approach to bridging the symbolic / subsymbolic gap, via very tightly integrating DeSTIN with OpenCog. We don't claim that this is the only way to bridge the gap, but we do believe it is a viable way. Given the existing DeSTIN and OpenCog designs and codebases, the execution of the ideas outlined here seems to be relatively straightforward, falling closer to the category of "advanced development" than that of blue-sky research. However, fine-tuning all the details of the approach will surely require substantial effort.

While we have focused on robotics applications here, the basic ideas described could be implemented and evaluated in a variety of other contexts as well, for example the identification of objects and events in videos, or

intelligent video summarization. Our interests are broad, however, we feel that robotics is the best place to start – partly due to a general intuition regarding the deep coupling between human-like intelligence and human-like embodiment; and partly due to a more specific intuition regarding the value of action for perception, as reflected in Heinz von Foerster's dictum "if you want to see, learn how to act". We suspect there are important cognitive reasons why perception in the human brain centrally involves premotor regions. The coupling of a perceptual deep learning hierarchy and a symbolic AI system doesn't intrinsically solve the combinatorial explosion problem intrinsic in looking for potential conceptual patterns in masses of perceptual data. However, a system with particular goals and the desire to act in such a way as to achieve them, possesses a very natural heuristic for pruning the space of possible perceptual/conceptual patterns. It allows the mind to focus in on those percepts and concepts that are useful for action. Of course, there are other ways besides integrating action to enforce effective pruning, but the integration of perception and action has a variety of desirable properties that might be difficult to emulate via other methods, such as the natural alignment of the hierarchical structures of action and reward with that of perception.

The outcome of any complex research project is difficult to foresee in detail. However, our intuition – based on our experience with OpenCog and DeSTIN, and our work with the mathematical and conceptual theories underlying these two systems – is that the hybridization of OpenCog and DeSTIN as described here will constitute a major step along the path to human-level AGI. It will enable the creation of an OpenCog instance endowed with the capability of flexibly interacting with a rich stream of data from the everyday human world. This data will not only help OpenCog to guide a robot in carrying out everyday tasks, but will also provide raw material for OpenCogPrimes cognitive processes to generalize from in various ways – e.g. to use as the basis for the formation of new concepts or analogical inferences.

Section IX
Procedure Learning

Chapter 30
Procedure Learning as Program Learning

30.1 Introduction

Broadly speaking, the learning of predicates and schemata (executable procedures) is done in CogPrime via a number of different methods, including for example PLN inference and concept predicatization (to be discussed in later chapters). Most of these methods, however, merely extrapolate procedures directly from other procedures or concepts in the AtomSpace, in a *local* way – a new procedure is derived from a small number of other procedures or concepts. General intelligence also requires a method for deriving new procedures that are more "fundamentally new." This is where CogPrime makes recourse to explicit procedure learning algorithms such as hillclimbing and MOSES, discussed in Chapters 32 and 33 below.

In this brief chapter we formulate the procedure learning problem as a program learning problem in a general way, and make some high-level observations about it. Conceptually, this chapter is a follow-up to Chapter 21, which discussed the choice to represent procedures as programs; here we make some simple observations regarding the implications of this choice for procedure learning, and the formal representation of procedure learning with OpenCog.

30.1.1 Program Learning

An optimization problem may be defined as follows: a solution space $\boldsymbol{S}$ is specified, together with some fitness function on solutions, where "solving the problem" corresponds to discovering a solution in $\boldsymbol{S}$ with a sufficiently high fitness.

In this context, we may define **program learning** as follows: given a program space $\boldsymbol{P}$, a behavior space $\boldsymbol{B}$, an execution function $\boldsymbol{exec : P \mapsto B}$,

and a fitness function on *behaviors*, "solving the problem" corresponds to discovering a program $\boldsymbol{p}$ in $\boldsymbol{P}$ whose corresponding behavior, $\boldsymbol{exec(p)}$, has a sufficiently high fitness.

In evolutionary learning terms, the program space is the space of **genotypes**, and the behavior space is the space of **phenotypes**.

This formalism of procedure learning serves well for explicit procedure learning CogPrime , not counting cases like procedure learning within other systems (like DeSTIN) that may be hybridized with CogPrime .

Of course, this extended formalism can of be entirely vacuous – the behavior space could be identical to the program space, and the execution function simply identity, allowing any optimization problem to be cast as a problem of program learning. The utility of this specification arises when we make interesting assumptions regarding the program and behavior spaces, and the execution and fitness functions (thus incorporating additional inductive bias):

1. **Open-endedness** – $\boldsymbol{P}$ has a natural "program size" measure – programs may be enumerated from smallest to largest, and there is no obvious problem-independent upper bound on program size.
2. **Over-representation** – $\boldsymbol{exec}$ often maps many programs to the same behavior.
3. **Compositional hierarchy** – programs themselves have an intrinsic hierarchical organization, and may contain subprograms which are themselves members of $\boldsymbol{P}$ or some related program space. This provides a natural family of distance measures on programs, in terms of the the number and type of compositions / decompositions needed to transform one program into another (i.e., edit distance).
4. **Chaotic Execution** – very similar programs (as conceptualized in the previous item) may have very different behaviors.

Precise mathematical definitions could be given for all of these properties but would provide little insight – it is more instructive to simply note their ubiquity in symbolic representations; human programming languages (LISP, C, etc.), Boolean and real-valued formulae, pattern-matching systems, automata, and many more. The crux of this line of thought is that the combination of these four factors conspires to *scramble* fitness functions – even if the mapping from behaviors to fitnesss is separable or nearly decomposable, the complex* program space and chaotic execution function will often quickly lead to intractability as problem size grows. These properties are not superficial inconveniences that can be circumvented by some particularly clever encoding. On the contrary, they are the essential characteristics that give programs the power to compress knowledge and generalize correctly, in contrast to flat, inert representations such as lookup tables (see Baum [Bau04] for a full treatment of this line of argument).

---

* Here "complex" means open-ended, over-representing, and hierarchical.

The consequences of this particular kind of complexity, together with the fact that most program spaces of interest are combinatorially very large, might lead one to believe that competent program learning is impossible. Not so: real-world program learning tasks of interest have a compact structure[†] – they are not "needle in haystack" problems or uncorrelated fitness landscapes, although they can certainly be encoded as such. The most one can definitively state is that algorithm ***foo***, methodology ***bar,*** or representation ***baz*** is unsuitable for expressing and exploiting the regularities that occur across interesting program spaces. Some of these regularities are as follows:

1. **Simplicity prior** – our prior assigns greater probability mass to smaller programs.
2. **Simplicity preference** – given two programs mapping to the same behavior, we prefer the smaller program (this can be seen as a secondary fitness function).
3. **Behavioral decomposability** – the mapping between behaviors and fitness is separable or nearly decomposable. Relatedly, fitnesss are more than scalars – there is a partial ordering corresponding to behavioral dominance, where one behavior dominates another if it exhibits a strict superset of the latter's desideratum, according to the fitness function.[‡] This partial order will never contradict the total ordering of scalar fitnesss.
4. **White box execution** – the mechanism of program execution is known *a priori*, and remains constant across many problems.

How these regularities may be exploited will be discussed in later sections and chapters. Another fundamental regularity of great interest for artificial general intelligence is patterns across related problems that may be solvable with similar programs (e.g., involving common modules).

30.2 Representation-Building

One important issue in achieving competent program learning is *representation building*. In an ideally encoded optimization problem, all prespecified variables would exhibit complete *separability*, and could be optimized independently. Problems with hierarchical dependency structure cannot be encoded this way, but are still tractable by dynamically learning the problem decomposition (as is done by the BOA and hBOA, described in Chapter 33).

[†] Otherwise, humans could not write programs significantly more compact than lookup tables.

[‡] For example, in supervised classification one rule dominates another if it correctly classifies all of the items that second rule classifies correctly, as well as some which the second rule gets wrong.

For complex problems with interacting subcomponents, finding an accurate problem decomposition is often tantamount to finding a solution. In an idealized run of a competent optimization algorithm, the problem decomposition evolves along with the set of solutions being considered, with parallel convergence to the correct decomposition and the global solution optima. However, this is certainly contingent on the existence of some compact[§] and reasonably correct decomposition in the space (of decompositions, not solutions) being searched.

Difficulty arises when no such decomposition exists, or when a more effective decomposition exists that cannot be formulated as a probabilistic model over representational parameters. Accordingly, one may extend current approaches via either: a more general modeling language for expressing problem decompositions; or *additional mechanisms* that modify the representations on which modeling operates (introducing additional inductive bias). in CogPrime we have focused on the latter – the former would appear to require qualitatively more computational capacity than will be available in the near future. If one ignores this constraint, such a "universal" approach to general problem-solving is indeed possible, e.g. $AIXI^{tl}$ as discussed above.

We refer to these additional mechanisms as "representation-building" because they serve the same purpose as the pre-representational mechanisms employed (typically by humans) in setting up an optimization problem – to present an optimization algorithm with the salient parameters needed to build effective problem decompositions and vary solutions along meaningful dimensions. We return to this issue in detail in Chapter 33 in the context of MOSES, the most powerful procedure-learning algorithm provided in CogPrime .

30.3 Specification Based Procedure Learning

Now we explain how procedure learning fits in with the declarative and intentional knowledge representation in the Atomspace.

The basic method that CogPrime uses to learn procedures that appear fundamentally new from the point of view of the AtomSpace at a given point in time is "specification-based procedure learning". This involves taking a PredicateNode with a ProcedureNode input type as a specification, and searching for ProcedureNodes that fulfill this specification (in the sense of making the specification PredicateNode as true as possible). In evolutionary computing lingo, the specification predicate is a *fitness function*.

[§] The decomposition must be compact because in practice only a fairly small sampling of solutions may be evaluated (relative to the size of the total space) at a time, and the search mechanism for exploring decomposition-space is greedy and local. This is in also accordance with the general notion of learning corresponding to compression.

Searching for PredicateNodes that embody patterns in the AtomSpace as a whole is a special case of this kind of learning, where the specification PredicateNode embodies a notion of what constitutes an "interesting pattern". The quantification of interestingness is of course an interesting and nontrivial topic in itself.

Finding schemata that are likely to achieve goals important to the system is also a special case of this kind of learning. In this case, the specification predicate is of the form:

```
F(S) = PredictiveImplicationLink (ExOutLink S) G
```

This measures the extent to which executing schema S is positively correlated with goal-predicate G being achieved shortly later.

Given a PredicateNode interpretable as a specification, how do we find a ProcedureNode satisfying the specification? Lacking prior knowledge sufficient to enable an incremental approach like inference, we must search the space of possible ProcedureNodes, using an appropriate search heuristic, hopefully one that makes use of the system's existing knowledge as fully as possible.

Chapter 31
Learning Procedures via Imitation, Reinforcement and Correction

Co-authored with Moshe Looks, Samir Araujo and Welter Silva

31.1 Introduction

In procedure learning as elsewhere in cognition, it's not enough to use the right algorithm, one has to use it in the right way based on the data and context and affordances available. While Chapters **??** and 33 focus on procedure learning algorithms, this one focuses on procedure learning *methodology*. We will delve into the important special case of procedure learning in which the fitness function involves reinforcement and imitation supplied by a teacher and/or an environment, and look at examples of this in the context of teaching behaviors to virtual pets controlled by OpenCogPrime . While this may seem a very narrow context, many of the lessons learned are applicable more broadly; and the discussion has the advantage of being grounded in actual experiments done with OpenCogPrimes predecessor system, the Novamente Cognition Engine, and with an early OpenCog version as well, during the period 2007-2008.

We will focus mainly on learning from a teacher, and then common on the very similar case where the environment, rather than some specific agent, is the teacher.

31.2 IRC Learning

Suppose one intelligent agent (the "teacher") has knowledge of how to carry out a certain behavior, and wants to transfer this knowledge to another intelligent agent (the "student"). But, suppose the student agent lacks the power of language (which might be, for example, because language is the thing being taught!). How may the knowledge be transferred? At least three methodologies are possible:

1. **Imitative learning**: The teacher acts out the behavior, showing the student by example
2. **Reinforcement learning**: The student tries to do the behavior himself, and the teacher gives him feedback on how well he did
3. **Corrective learning**: As the student attempts the behavior, the teacher actively corrects (i.e. changes) the student's actions, guiding him toward correct performance

Obviously, these three forms of instruction are not exclusive. What we describe here, and call IRC learning, is a pragmatic methodology for instructing AGI systems that combines these three forms of instruction. We believe this combination is a potent one, and is certainly implicit in the way human beings typically teach young children and animals.

For sake of concreteness, we present IRC learning here primarily in the context of virtually embodied AGI systems – i.e., AGI systems that control virtual agents living in virtual worlds. There is an obvious extension to physical robots living in the real world and capable of flexible interaction with humans. In principle, IRC learning is applicable more broadly as well, and could be explored in various non-embodied context such as (for instance) automated theorem-proving. In general, the term "IRC learning" may be used to describe any teacher/student interaction that involves a combination of reinforcement, imitation and correction. While we have focused in our practical work so far on the use of IRC to teach simple "animal-like" behaviors, the application that interests us more in the medium term is language instruction, to which we will return in later chapters.

Harking back to Chapter 9, it is clear that an orientation toward effective IRC learning will be valuable for any system attempting to achieve complex goals in an environment heavily populated by other intelligences possessing significant goal-relevant knowledge. Everyday human environments possess this characteristic, and we suggest the best way to create human-level AGIs will be to allow them to develop in environments possessing this characteristic as well.

31.2.1 A Simple Example of Imitation/Reinforcement Learning

Perhaps the best way to introduce the essential nature of the IRC teaching protocol is to give a brief snippet from a script that was created to guide the actual training of the virtual animals controlled by the PetBrain. This snippet involves only I and R; the C will be discussed afterwards.

This snippet demonstrates a teaching methodology that involves two avatars: Bob who is being the teacher, and Jill who is being an "imitation animal," showing the animal what to do by example.

1. *Bob wants to teach the dog Fido a trick. He calls his friend Jill over. "Jill, can you help me teach Fido a trick?"*
2. *Jill comes over. "How much will you pay me for it?*
3. *Bob gives her a kiss.*
4. *"All right," says Jill, "what do you want to teach him?"*
5. *"Let's start with fetching stuff," replies Bob.*
6. *So Bob and Jill start teaching Fido to fetch using the Pet language....*
7. *Bob says: "Fido, I'm going to teach you to play fetch with Jill."*
8. *Fido sits attentively, looking at Bob.*
9. *Bob says: "OK, I'm playing fetch now."*
10. *Bob picks up a stick from the ground and throws it. Jill runs to get the stick and brings it back to Bob.*
11. *Bob says: "I'm done fetching.*
12. *Bob says, "You try it."*
13. *Bob throws a stick. Fido runs to the stick, gets it, and brings it back.*
14. *Bob says "Good dog!"*
15. *Fido looks happy.*
16. *Bob says: "Ok, we're done with that game of fetch.*
17. *Bob says, "Now, let's try playing fetch again."*
18. *This time, Bob throws a stick in a different direction, where there's already a stick lying on the ground (call the other stick Stick 2).*
19. *Fido runs and retrieves Stick 2. As soon as he picks it up, Bob says "No." But Fido keeps on running and brings the stick back to Bob.*
20. *Bob says "No, that was wrong. That was the wrong stick. Stop trying!"*
21. *Jill says, "Furry little moron!"*
22. *Bob says to Jill, "Have some patience, will you? Let's try again."*
23. *Fido is slowly wandering around, sniffing the ground.*
24. *Bob says "Fido, stay." Fido returns near Bob and sits.*
25. *Bob throws Stick 2. Fido starts to get up and Bob repeats "Fido, stay."*
26. *Bob goes and picks up Stick 1, and walks back to his original position.*
27. *Bob says "Fido, I'm playing fetch with Jill again."*
28. *Bob throws the first stick in the direction of stick 2.*
29. *Jill goes and gets stick 1 and brings it back to Bob.*
30. *Bob says "I'm done playing fetch with Jill."*
31. *Bob says "Try playing fetch with me now." He throws stick 1 in another direction, where stick 3 and stick 4 are lying on the ground, along with some other junk.*
32. *Fido runs and gets stick 1 and brings it back.*
33. *Bob and Jill both jump up and down smiling and say "Good dog! Good dog, Fido!! Good dog!!"*
34. *Fido smiles and jumps up and licks Jill on the face.*
35. *Bob says, "Fido, we're done practicing fetch."*

In the above transcript, Line 7 initiates a formal training session, and Line 33 terminates this session. The training session is broken into "exemplar" intervals during which exemplars are being given, and "trial" intervals during

which the animal is trying to imitate the exemplars, following which is receives reinforcement on its success or otherwise. For instance line 9 initiates the presentation of an exemplar interval, and line 11 indicates the termination of this interval. Line 12 indicates the beginning of a trial interval, and line 16 indicates the termination of this interval.

The above example of combined imitative/reinforcement learning involves two teachers, but, this is of course not the only way things can be done. Jill could be eliminated from the above teaching example. The result of this would be that, in figuring out how to imitate the exemplars, Fido would have to figure out which of Bob's actions were "teacher" actions and which were "simulated student" actions. This is not a particularly hard problem, but it's harder than the case where Jill carries out all the simulated-student actions. So in the case of teaching fetch with only one teacher avatar, on average, more reinforcement trials will be required.

31.2.2 A Simple Example of Corrective Learning

Another interesting twist on the imitative/reinforcement teaching methodology described above is the use of explicit correctional instructions from the teacher to the animal. This is not shown in the above example but represents an important addition to the methodology show there. One good example of the use of corrections would be the problem of teaching would be teaching an animal to sit and wait until the teacher says "Get Up," using only a single teacher. Obviously, using two teachers, this is a much easier problem. Using only one teacher, it's still easy, but involves a little more subtlety, and becomes much more tractable when corrections are allowed.

One way that human dog owners teach their dogs this sort of behavior is as follows:

1. Tell the dog "sit"
2. tell the dog "stay"
3. Whenever the dog tries to get up, tell him "no" or "sit", and then he sits down again
4. eventually, tell the dog to "get up"

The real dog understands, in its own way, that the "no" and "sit" commands said after the "stay" command are meta-commands rather than part of the "stay" behavior.

In our virtual-pet case, this would be more like

1. tell the dog "I'm teaching you to stay"
2. Tell the dog "sit"
3. Whenever the dog tries to get up, tell him "no" or "sit", and then he sits down again

4. eventually, tell the dog to "get up"
5. tell the dog "I'm done teaching you to stay"

One easy way to do this, which deviates from the pattern of humanlike interaction, would be to give the agent knowledge about how to interpret an explicit META flag in communications directed toward it. In this case, the teaching would look like

1. tell the dog "I'm teaching you to stay"
2. Tell the dog "META: sit"
3. Whenever the dog tries to get up, tell him "META: no" or "META:sit", and then he sits down again
4. eventually, tell the dog to "get up"
5. tell the dog "I'm done teaching you to stay"–

Even without the META tag, this behavior (and other comparable ones) is learnable via CogPrime 's learning algorithms within a modest number of reinforcement trials. So we have not actually implemented the META approach. But it well illustrates the give-and-take relationship between the sophistication of the teaching methodology and the number of reinforcement trials required. In many cases, the best way to reduce the number of reinforcement trials required to learn a behavior is not to increase the sophistication of the learning algorithm, but rather to increase the information provided during the instruction process. No matter how advanced the learning algorithm, if the teaching methodology only gives a small amount of information, it's going to take a bunch of reinforcement trials to go through the search space and find one of the right procedures satisfying the teacher's desires. One of the differences between the real-world learning that an animal or human child (or adult) experiences, and the learning "experienced" by standard machine-learning algorithms, is the richness and diversity of information that the real world teaching environment provides, beyond simple reinforcement signals. Virtual worlds provide a natural venue in which to experiment with providing this sort of richer feedback to AI learning systems, which is one among the many reasons why we feel that virtual worlds are an excellent venue for experimentation with and education of early-stage AGI systems.

31.3 IRC Learning in the PetBrain

Continuing the theme of the previous section, we now discuss "trick learning" in the PetBrain, as tested using OpenCog and the Multiverse virtual world during 2007-2008. The PetBrain constitutes a specific cognitive infrastructure implementing the IRC learning methodology in the virtual-animal context, with some extensibility beyond this context as well.

In the PetBrain, learning itself is carried out by a variety of hillclimbing as described in Chapter 32, which is a fast learning algorithm but may fail

on harder behaviors (in the sense of requiring an unacceptably large number of reinforcement trials). For more complex behaviors, MOSES (Chapter 33) would need to be integrated as an alternative. Compared to hillclimbing, MOSES is much smarter but slower, and may take a few minutes to solve a problem. The two algorithms (as implemented for the PetBrain) share the same Combo knowledge representation and some other software components (e.g. normalization rules for placing procedures in an appropriate hierarchical normal form, as described in Chapter 21).

The big challenge involved in designing the PetBrain system, AI-wise, was that these learning algorithms, used in a straightforward way with feedback from a human-controlled avatar as the fitness function, would have needed need an excessive number of reinforcement trials to learn relatively simple behaviors. This would bore the human beings involved with teaching the animals. This is not a flaw of the particular learning algorithms being proposed, but is a generic problem that would exist with any AI algorithms. To choose an appropriate behavior out of the space of all possible behaviors satisfying reasonable constraints, requires more bits of information that is contained in a handful of reinforcement trials.

Most "animal training" games (e.g. Nintendogs may be considered as a reference case) work around this "hard problem" by not allowing teaching of novel behaviors. Instead, a behavior list is made up front by the game designers. The animals have preprogrammed procedures for carrying out the behaviors on the list. As training proceeds they make fewer errors, till after enough training they converge "miraculously" on the pre-programmed plan

This approach only works, however, if all the behaviors the animals will ever learn have been planned and scripted in advance.

The first key to making learning of non-pre-programmed behaviors work, without an excessive number of reinforcement trials, is in "fitness estimation" – code that guesses the fitness of a candidate procedure at fulfilling the teacher's definition of a certain behavior, without actually having to try out the procedure and see how it works. This is where the I part of IRC learning comes in.

At an early stage in designing the PetBrain application, we realized it would be best if the animals were instructed via a methodology where the same behaviors are defined by the teacher both by demonstration *and* by reinforcement signals. Learning based on reinforcement signals only can also be handled, but learning will be much slower.

In evolutionary programming lingo, we have

1. Procedures = genotypes
2. Demonstrated exemplars, and behaviors generated via procedures = phenotypes
3. Reinforcement signals from pet owner = fitness

One method of imitation-based fitness estimation used in the PetBrain involves an internal simulation world which we'll call CogSim, as discussed in

Chapter 40 *. CogSim can be visualized using a simple testing UI, but in the normal course of operations it doesn't require a user interface; it is an internal simulation world, which allows the PetBrain to experiment and see what a certain procedure would be likely to do if enacted in the SL virtual world. Of course, the accuracy of this kind of simulation depends on the nature of the procedure. For procedures that solely involve moving around and interacting with inanimate objects, it can be very effective. For procedures involving interaction with human-controlled avatars, other animals, or other complex objects, it may be unreliable – and making it even moderately reliable would require significant work that has not yet been done, in terms of endowing CogSim with realistic simulations of other agents and their internal motivational structures and so forth. But short of this, CogSim has nonetheless proved useful for estimating the fitness of simple behavioral procedures.

When a procedure is enacted in CogSim, this produces an object called a "behavior description" (BD), which is represented in the AtomSpace knowledge representation format. The BD generated by the procedure is then compared with the BD's corresponding to the "exemplar" behaviors that the teacher has generated, and that the student is trying to emulate. Similarities are calculated, which is a fairly subtle matter that involves some heuristic inferences. An estimate of the likelihood that the procedure, if executed in the world, will generate a behavior adequately similar to the exemplar behaviors.

Furthermore, this process of estimation may be extended to make use of the animal's long-term episodic memory. Suppose a procedure P is being evaluated in the context of exemplar-set E. Then

1. The episodic memory is mined for pairs (P', E') that are similar to (P,E)
2. The fitness of these pairs (P', E') is gathered from the experience base
3. An estimate of the fitness of (P,E) is then formed

Of course, if a behavior description corresponding to P has been generated via CogSim, this may also be used in the similarity matching against long-term memory. The tricky part here, of course, is the similarity measurement itself, which can be handled via simple heuristics, but if taken sufficiently seriously becomes a complex problem of uncertain inference.

One thing to note here is that in the PetBrain context, although learning is done by each animal individually, this learning is subtly guided by collective knowledge within the fitness estimation process. Internally, we have a "borg mind" with multiple animal bodies, and an architecture designed to ensure the maintenance of unique personalities on the part of the individual animals in spite of the collective knowledge and learning underneath.

At time of writing, we have just begun to experiment with the learning system as described above, and are using it to learn simple behaviors such as playing fetch, basic soccer skills, doing specific dances as demonstrated by the

* The few readers familiar with obscure OpenCog documentation may remember that CogSim was previously called "Third Life", in reference to the Second Life virtual world that was being used to embody the OpenCog virtual pets at the time

teacher, and so forth. We have not yet done enough experimentation to get a solid feel for the limitations of the methodology as currently implemented.

Note also the possibility of using CogPrime 's PLN inference component to allow generalization of learned behaviors. For instance, with inference deployed appropriately, a pet that had learned how to play tag would afterwards have a relatively easy time learning to play "freeze tag." A pet that had learned how to hunt for Easter eggs would have a relatively easy time learning to play hide-and-seek. Episodic memory can be very useful for fitness estimation here, but explicit use of inference may allow much more rapid and far-reaching inference capabilities.

31.3.1 Introducing Corrective Learning

Next, how may corrections be utilized in the learning process we have described? Obviously, the corrected behavior description gets added into the knowledge base as an additional exemplar. And, the fact of the correction acts as a partial reinforcement (up until the time of the correction, what the animal was doing was correct). But beyond this, what's necessary is to propagate the correction backward from the BD level to the procedure level. For instance, if the animal is supposed to be staying in one place, and it starts to get up but is corrected by the teacher (who says "sit" or physically pushes the animal back down), then the part of the behavior-generating procedure that directly generated the "sit" command needs to be "punished." How difficult this is to do, depends on how complex the procedure is. It may be as simple as providing a negative reinforcement to a specific "program tree node" within the procedure, thus disincentivizing future procedures generated by the procedure learning algorithm from containing this node. Or it may be more complex, requiring the solution of an inference problem of the form "Find a procedure P" that is as similar as possible to procedure P, but that does not generate the corrected behavior, but rather generates the behavior that the teacher wanted instead." This sort of "working backwards from the behavior description to the procedure" is never going to be perfect except in extremely simple cases, but it is an important part of learning. We have not yet experimented with this extensively in our virtual animals, but plan to do so as the project proceeds.

There is also an interesting variant of correction in which the agent's own memory serves implicitly as the teacher. That is, if a procedure generates a behavior that seems wrong based on the history of successful behavior descriptions for similar exemplars, then the system may suppress that particular behavior or replace it with another one that seems more appropriate – inference based on history thus serving the role of a correcting teacher.

31.4 Applying A Similar IRC Methodology to Spontaneous Learning

We have described the IRC teaching/learning methodology in the context of learning from a teacher – but in fact a similar approach can be utilized for purely unsupervised learning. In that case, the animal's intrinsic goal system acts implicitly as a teacher.

For instance, suppose the animal wants to learn how to better get itself fed. In this case,

1. Exemplars are provided by instances in the animal's history when it has successfully gotten itself fed
2. Reinforcement is provided by, when it is executing a certain procedure, whether or not it actually gets itself fed or not
3. Correction as such doesn't apply, but implicit correction may be used via deploying history-based inference. If a procedure generates a behavior that seems wrong based on the history of successful behavior descriptions for the goal of getting fed, then the system may suppress that particular behavior.

The only real added complexity here lies in identifying the exemplars. In surveying its own history, the animal must look at each previous instance in which it got fed (or same sample thereof), and for each one recollect the series of N actions that it carried out prior to getting fed. It then must figure out how to set N – i.e. which of the actions prior to getting fed were part of the behavior that led up to getting fed, and which were just other things the animal happened to be doing a while before getting fed. To the extent that this exemplar mining problem can be solved adequately, innate-goal-directed spontaneous learning becomes closely analogous to teacher-driven learning as we've described it. Or in other words: Experience, as is well known, can serve as a very effective teacher.

Chapter 32
Procedure Learning via Adaptively Biased Hillclimbing

Primary author: Nil Geisweiller

32.1 Introduction

Having chosen to represent procedures as programs, explicit procedure learning then becomes a matter of automated program learning. In its most general incarnation, automated program learning is obviously an intractable problem; so the procedure learning design problem then boils down to finding procedure learning algorithms that are effective on the class of problems relevant to CogPrime systems in practice. This is a subtle matter because there is no straightforward way to map from the vaguely-defined category of real-world "everyday human world like" goals and environments to any formally-defined class of relevant objective functions for a program learning algorithm.

However, this difficulty is not a particular artifact of the choice of programs to represent procedures; similar issues would arise with any known representational mechanism of suitable power. For instance, if procedures were represented as recurrent neural nets, there would arise similar questions of how many layers to give the networks, how to determine the connection statistics, what sorts of neurons to use, which learning algorithm, etc. One can always push such problems to the meta level and use automated learning to determine which variety of learning algorithm to use – but then one has to make some decisions on the metalearning level, based on one's understanding of the specific structure of the space of relevant program learning algorithms. In the fictitious work of unbounded computational resources no such judicious choices are necessary, but that's not the world we live in, and it's not relevant to the design of human-like AGI systems.

At the moment, in CogPrime , we utilize two different procedure learning systems, which operate on the same knowledge representation and rely on much of the same internal code. One, which we roughly label "hill climbing", is used for problems that are sufficiently "easy" in the sense that it's possible for the system to solve them using feasible resources without (implicitly or explicitly) building any kind of sophisticated model of the space of solutions

to the problem. The other, MOSES, is used for problems that are sufficiently difficult that the right way to solve them is to build a model of the program space, progressively as one tries out various solutions, and then use this model to guide ongoing search for better and better solutions. Hillclimbing is treated in this chapter; MOSES in the next.

32.2 Hillclimbing

"Hillclimbing," broadly speaking, is not a specific algorithm but a category of algorithms. It applies in general to any search problem where there is a large space of possible solutions, which can be compared as to their solution quality. Here we are interested in applying it specifically to problems of search through spaces of *programs*.

In hillclimbing, one starts with a candidate solution to a problem (often a random one, which may be very low-quality), and iteratively makes small changes to the candidate to generate new possibilities, hoping one of them will be a better solution. If a new possibility is better than the current candidate, then the algorithm adopts the new possibility as its new candidate solution. When the current candidate solution can no longer be improved via small changes, the algorithm terminates. Ideally, at that point the current candidate solution is close to optimal – but this is not guaranteed!

Various tweaks to hillclimbing exist, including "restart" which means that one starts hillclimbing over and over again, taking the best solution from multiple trials; and "backtracking", which means that if the algorithm terminates at a solution that seems not adequate, then the search can "backtrack" to a previously considered candidate solution, and try to make different small changes to that candidate solution, trying previously unexplored possibilities in search of a new candidate. The value of these and other tweaks depends on the specific problem under consideration.

In the specific approach to hillclimbing described here, we use a hillclimber with backtracking, applied to programs that are represented in the same hierarchical normal form used with MOSES (based on the program normalization ideas presented in Chapter 21). The basic pseudocode for the hillclimber may be given as:

Let L be the list (initially empty) of programs explored so far in decreasing order with respect to their fitness. Let N_p be the neighbors of program p.

1. Take the best program $p \in L$
2. Evaluate all programs of N_p
3. Merge N_p in L
4. Move p from L to the set of best programs found so far and repeat from step 1 until time runs out

In the following sections of this chapter, we show how to speed up the hillclimbing search for learning procedures via four optimizations, which have been tested fairly extensively. For concreteness we will refer often to the specific case of using the hill climbing algorithm to control a virtual agent in a virtual world – and especially the case of teaching a virtual pet tricks via imitation learning (as in Chapter 31) but the ideas have more general importance. The four optimizations are:

- reduce candidates to normal form to minimize over-representation and increase the syntactic semantic correlation (Chapter 21),
- filter perceptions using an entropy criterion to avoid building candidates that involve nodes unlikely to be contained in the solution (Section 32.3),
- use sequences of agent actions, observed during the execution of the program, as building blocks (Section 32.4),
- choose and calibrate a simplicity measure to focus on simpler solutions (the "Occam bias") first (Section 32.5).

32.3 Entity and Perception Filters

The number of program candidates of a given size increases exponentially with the alphabet of the language; therefore it is important to narrow that alphabet as much as possible. This is the role of the two filters explained below, the Entity and the Entropy filter.

32.3.1 Entity filter

This filter is in charge of selecting the entities in the scene the pet should take into account during an imitation learning session. These entities can be any objects, avatars or other pets.

In general this is a very hard problem, for instance if a bird is flying near the owner while teaching a trick, should the pet ignore it? Perhaps the owner wants to teach the pet to bark at them; if so they should not be ignored.

In our current and prior work with OpenCog controlling virtual world agents, we have used some fairly crude heuristics for entity filtering, which must be hand-tuned depending on the properties of the virtual world. However, our intention is to replace these heuristics with entity filtering based on Economic Attention Networks (ECAN) as described in Chapter 23.

32.3.2 Entropy perception filter

The perception filter is in charge of selecting all perceptions in the scene that are reasonably likely to be part of the solution to the program learning problem posed. A "perception" in the virtual world context means the evaluation of one of a set of pre-specified perception predicates, with an argument consisting of one of the entities in the observed environment.

Given N entities (provided by the Entity filter), there are usually $O(N^2)$ potential perceptions in the Atomspace due to binary perceptions like

```
near(owner bird)
inside(toy box)
...
```

The perception filter proceeds by computing the entropy of any potential perceptions happening during a learning session. Indeed if the entropy of a given perception `P` is high that means that a conditional `if(P B1 B2)` has a rather balanced probability of taking Branch `B1` or `B2`. On the other hand if the entropy is low then the probability of taking these branches is unbalanced, for instance the probability of taking `B1` may be significantly higher than the probability of taking `B2`, and therefore `if(P B1 B2)` could reasonably be substituted by `B1`.

For example, assume that during the teaching sessions, the predicate `near(owner bird)` is false 99% percent of the time; then `near(owner bird)` will have a low entropy and will possibly be discarded by the filter (depending on the threshold). If the bird is always far from the owner then it will have entropy 0 and will surely be discarded, but if the bird comes and goes it will have a high entropy and will pass the filter. Let P be such a perception and P_t returns 1 when the perception is true at time t or 0 otherwise, where t ranges over the set of instants, of size N, recorded between the beginning and the end of the demonstrated trick. The calculation goes as follows

$$Entropy(P) = H\left(\frac{\sum_t P_t}{N}\right)$$

where $H(p) = -p\ log(p) - (1-p)log(1-p)$. There are additional subtleties when the perception involves random operators, like `near(owner random_object)` that is the entropy is calculated by taking into account a certain distribution over entities grouped under the term random_object. The calculation is optimized to ignore instants when the perception relates to object that have not moved which makes the calculation efficient enough, but there is room to improve it in various ways; for instance it could be made to choose perceptions based not only on entropy but also inferred relevancy with respect to the context using PLN.

32.4 Using Action Sequences as Building Blocks

A heuristic that has been shown to work, in the "virtual pet trick" context, is to consider sequences of actions that are compatible with the behavior demonstrated by the avatar showing the trick as building blocks when defining the neighborhood of a candidate. For instance if the trick is to fetch a ball, compatible sequences would be

`goto(ball), grab(ball), goto(owner), drop`
`goto(random_object), grab(nearest_object), goto(owner), drop`
...

Sub-sequences can be considered as well – though too many building blocks also increase the neighborhood exponentially, so one has to be careful when doing that. In practice using the set of whole compatible sequences worked well. This for instance can speed up many fold the learning of the trick triple_kick as shown in Section 32.6.

32.5 Automatically Parametrizing the Program Size Penalty

A common heuristic for program learning is an "Occam penalty" that penalizes large programs, hence biasing search toward compact programs. The function we use to penalize program size is inspired by Ray Solomonoff's theory of optimal inductive inference [Sol64a, Sol64b]; simply said, a program is penalized exponentially with respect to its size. Also, one may say that since the number of program candidates grows exponentially with their size, exploring solutions with higher size must be exponentially worth the cost.

In the next subsections we describe the particular penalty function we have used and how to tune its parameters.

32.5.1 Definition of the complexity penalty

Let p be a program candidate and $penalty(p)$ a function with domain [0,1] measuring the complexity of p. If we consider the complexity penalty function $penalty(p)$ as if it denotes the prior probability of p, and $score(p)$ (the quality of p as utilized within the hill climbing algorithm) as denoting the conditional probability of the desired behavior knowing p, then Bayes rule* tells us that

* Bayes rule as used here is $P(M|D) = \frac{P(M)P(D|M)}{P(D)}$ where M denotes the Model (the program) and D denotes the data (the behavior to imitate), here $P(D)$ is ignored, that is the data is assumed to be distributed uniformly

$$fitness(p) = score(p) \times penalty(p)$$

denotes the conditional probability of p knowing the right behavior to imitate, the fitness function that we want to maximize.

It happens that in the pet trick learning context which is our main example in this chapter, $score(p)$ does not denote such a probability; instead it measures how similar the behavior generated by p and the behavior to imitate are. However, we utilize the above formula anyway, with a heuristic interpretation. One may construct assumptions under which $score(p)$ does represent a probability but this would take us too far afield.

The penalty function we use is then given by:

$$penalty(p) = exp(-a \times log(b \times |A| + e) \times |p|)$$

where $|p|$ is the program size, $|A|$ its alphabet size and $e = exp(1)$. The reason $|A|$ enters into the equation is because the alphabet size varies from one problem to another due to the perception and action filters. Without that constraint the term $log(b \times |A| + e)$ could simply be included in a. The higher a the more intense the penalty is. The parameter b controls how that intensity varies with the alphabet size.

It is important to remark the difference between such a penalty function and lexicographic parsimony pressure (literally said *everything being equal, choose the shortest program*). Because of the use of sequences as building blocks, without such a penalty function the algorithm may rapidly reach an optimal program (a mere long sequence of actions) and remain stuck in an apparent optimum while missing the very logic of the action sequence that the human wants to convey.

32.5.2 Parameterizing the complexity penalty

Due to the nature of the search algorithm (hill climbing with restart), the choice of the candidate used to restart the search is crucial. In our case we restart with the candidate with the best fitness so far which has not been yet used to restart. The danger of such an approach is that when the algorithm enters a region with local optima (like a plateau), it may basically stay there as long as there exist better candidates in that region than outside of it non used yet for restart. Longer programs tend to generate larger regions of local optima (because they have exponentially more syntactic variations that lead to close behaviors), so if the search enters such region via an overly complex program it is likely to take a very long time to get out of it. Introducing probability in the choice of the restart may help avoiding that sort of trap but having experimented with that it turned out not to be significantly better on average for learning relatively simple things (indeed although the restart choice is more diverse it still tends to occur in large region of local optima).

However, a significant improvement we have found is to carefully choose the size penalty function so that the search will tend to restart on simpler programs even if they do not exhibit the best behaviors, but will still be able to reach the optimal solution even if it is a complex one.

The solution we suggest is to choose a and b such that $penalty(p)$ is:

1. as penalizing as possible, to focus on simpler programs first (although that constraint may possibly be lightened as the experimentation shows),
2. but still correct in the sense that the optimal solution p maximizes $fitness(p)$.

And we want that to work for all problems we are interested in. That restriction is an important point because it is likely that in general the second constraint will be too strict to produce a good penalty function.

We will now formalize the above problem. Let i be an index that ranges over the set of problems of interest (in our case pet tricks to learn), $score_i$ and $fitness_i$ denotes the score and fitness functions of the i^{th} problem. Let $\Theta_i(s)$ denote the set of programs of score s

$$\Theta_i(s) = \{p | score(p) = s\}$$

Define a family of partial functions

$$f_i : [0, 1] \mapsto \mathbb{N}$$

so that

$$f_i(s) = \underset{p \in \Theta_i(p)}{\operatorname{argmin}} |p|$$

What this says is that for any given score s we want the size of the shortest program p with that score. And f_i is partial because there may not be any program returning a given score.

Let be the family of partial functions

$$g_i : [0, 1] \mapsto [0, 1]$$

parametrized by a and b such that

$$g_i(s) = s \times exp(-a \times (log(b \times |A| + e) \times f_i(s))$$

That is: given a score s, $g_i(s)$ returns the fitness $fitness(p)$ of the shortest program p that marks that score.

32.5.3 Definition of the Optimization Problem

Let s_i be the highest score obtained for fitness function i (that is the score of the program chosen as the current best solution of i). Now the optimization problem consists of finding some a and b such that

$$\forall i \ \underset{s}{\text{argmax}}\ g_i(s) = s_i$$

that is the highest score has also the highest fitness. We started by choosing a and b as high as possible, it is a good heuristic but not the best, the best one would be to choose a and b so that they minimize the number of iterations (number of restarts) to reach a global optimum, which is a harder problem.

Also, regarding the resolution of the above equation, it is worth noting we do not need the analytical expression of $score(p)$. Using past learning experiences we can get a partial description of the fitness landscape of each problem just by looking at the traces of the search.

Overall we have found this optimization works rather well; that is, tricks that would otherwise take several hours or days of computation can be learned in seconds or minutes. And the method also enables fast learning for new tricks, in fact all tricks we have experimented with so far could be learned reasonably fast (seconds or minutes) without the need to retune the penalty function.

In the current CogPrime codebase, the algorithm in charge of calibrating the parameters of the penalty function has been written in Python. It takes in input the log of the imitation learning engine that contains the score, the size, the penalty and the fitness of all candidates explored for all tricks taken in consideration for the parameterizing. The algorithm proceeds in 2 steps:

1. Reconstitute the partial functions f_i for all fitness functions i already attempted, based on the traces of these previously optimized fitness functions.
2. Try to find the highest a and b so that

$$\forall i \ \underset{s}{\text{argmax}}\ g_i(s) = s_i$$

For step 2, since there are only two parameters to tune, we have used a 2D grid, enumerating all points (a, b) and zooming when necessary. So the speed of the process depends largely on the resolution of the grid but (on an ordinary 2009 PC processor) usually it does not require more than 20 minutes to both extract f_i and find a and b with a satisfactory resolution.

32.6 Some Simple Experimental Results

To test the above ideas in a simple context, we initially used them to enable an OpenCog powered virtual world agent to learn a variety of simple "dog tricks" based on imitation and reinforcement learning in the Multiverse virtual world. We have since deployed them on a variety of other applications in various domains.

We began these experiments by running learning on two tricks, fetch_ball and triple_kick to be described below, in order to calibrate the size penalty function:

1. fetch_ball, which corresponds to the Combo program

```
and_seq(goto(ball)
        grab(ball)
        goto(owner)
        drop)
```

2. triple_kick, if the stick is near the ball then kick 3 times with the left leg and otherwise 3 times with the right leg. So for that trick the owner had to provide 2 exemplars, one for kickL (with the stick near the ball) and one for kickR, and move away the ball from the stick before showing the second exemplar. Below is the Combo program of triple_kick

```
if(near(stick ball)
   and_seq(kickL kickL kickL)
   and_seq(kickR kickR kickR))
```

Before choosing an exponential size penalty function and calibrating it fetch_ball would be learned rather rapidly in a few seconds, but triple_kick would take more than an hour. After calibration both fetch_ball and triple_kick would be learned rapidly, the later in less than a minute.

Then we experimented with a new few tricks, some simpler, like sit_under_tree

```
and_seq(goto(tree) sit)
```

and others more complex like double_dance, where the trick consists of dancing until the owner emits the message "stop dancing", and changing the dance upon owner's actions

```
while(not(says(owner ``stop dancing''))
      if(last_action(owner ``kickL'')
         tap_dance
         lean_rock_dance))
```

That is the pet performs a tap_dance when the last action of the owner is kickL, and otherwise performs a lean_rock_dance.

We tested learning for 3 tricks, fetch_ball, triple_kick and double_dance. Each trick was tested in 7 settings denoted $conf_1$ to $conf_{10}$ summarized in Table 32.1.

- $conf_1$ is the default configuration of the system, the parameters of the size penalty function are $a = 0.03$ and $b = 0.34$, which is actually not what

is returned by the calibration technique but close to. That is because in practice we have found that in average learning is working slightly faster with these values.

- $conf_2$ is the configuration with the exact values returned by the calibration, that is $a = 0.05$, $b = 0.94$.
- $conf_3$ has the reduction engine disabled.
- $conf_4$ has the entropy filter disabled (threshold is null so all perceptions pass the filter).
- $conf_5$ has the intensity of the penalty function set to 0.
- $conf_6$ has the penalty function set with low intensity.
- $conf_7$ and $conf_8$ have the penalty function set with high intensity.
- $conf_9$ has the action sequence building block enabled
- $conf_{10}$ has the action sequence building block enabled but with a slightly lower intensity of the size penalty function than normal.

Reduct	ActSeq	Entropy	a	b	Setting
On	Off	0.1	0.03	0.34	$conf_1$
On	Off	0.1	0.05	0.94	$conf_2$
Off	Off	0.1	0.03	0.34	$conf_3$
On	Off	0	0.03	0.34	$conf_4$
On	Off	0.1	0	0.34	$conf_5$
On	Off	0.1	0.0003	0.34	$conf_6$
On	Off	0.1	0.3	0.34	$conf_7$
On	Off	0.1	3	0.34	$conf_8$
On	On	0.1	0.03	0.34	$conf_9$
On	On	0.1	0.025	0.34	$conf_{10}$

Table 32.1 Settings for each learning experiment

Setting	Percep	Restart	Eval	Time
$conf_1$	3	3	653	5s18
$conf_2$	3	3	245	2s
$conf_3$	3	3	1073	8s42
$conf_4$	136	3	28287	4mn7s
$conf_5$	3	>700	>500000	>1h
$conf_6$	3	3	653	5s18
$conf_7$	3	8	3121	23s42
$conf_8$	3	147	65948	8mn10s
$conf_9$	3	0	89	410ms
$conf_{10}$	3	0	33	161ms

Table 32.2 Learning time for fetch_ball

Tables 32.2, 32.3 and 32.4 contain the results of the learning experiment for the three tricks, fetch_ball, triple_kick and double_dance. In each table the column Percept gives the number perceptions which is taken into account

Setting	Percep	Restart	Eval	Time
$conf_1$	1	18	2783	21s47
$conf_2$	1	110	11426	1mn53s
$conf_3$	1	49	15069	2mn15s
$conf_4$	124	∞	∞	∞
$conf_5$	1	>800	>200K	>1h
$conf_6$	1	7	1191	9s67
$conf_7$	1	>2500	>200K	>1h
$conf_8$	1	>2500	>200K	>1h
$conf_9$	1	0	107	146ms
$conf_{10}$	1	0	101	164ms

Table 32.3 Learning time for triple_kick

Setting	Percep	Restart	Eval	Time
$conf_1$	5	1	113	4s
$conf_2$	5	1	113	4s
$conf_3$	5	1	150	6s20ms
$conf_4$	139	>4	>60K	>1h
$conf_5$	5	1	113	4s
$conf_6$	5	1	113	4s
$conf_7$	5	1	113	4s
$conf_8$	5	>1000	>300K	>1h
$conf_9$	5	1	138	4s191ms
$conf_{10}$	5	181	219K	56mn3s

Table 32.4 Learning time for double_dance

for the learning. Restart gives the number of restarts hill climbing had to do before reaching the solution. Eval gives the number of evaluations and Time the search time.

In Table 32.2 and 32.4 we can see that fetch_ball or double_dance are learned in a few seconds both in $conf_1$ and $conf_2$. In 32.3 however learning is about five times faster with $conf_1$ than with $conf_2$, which was the motivation to go with $conf_2$ as default configuration, but $conf_2$ still performs well.

As Tables 32.2, 32.3 and 32.4 demonstrate for setting $conf_3$, the reduction engine speeds the search up by less than twice for fetch_ball and double_dance, and many times for triple_kick.

The results for $conf_4$ shows the importance of the filtering function, learning is dramatically slowed down without it. A simple trick like fetch_ball took few minutes instead of seconds, double_dance could not be learned after an hour, and triple_kick might never be learned because it did not focus on the right perception from the start.

The results for $conf_5$ shows that without any kind of complexity penalty learning can be dramatically slowed down, for the reasons explained in *Section 32.5* , that is the search looses itself in large regions of sub-optima. Only double_dance was not affected by that, which is probably explained by the fact that only one restart occurred in double_dance and it happened to be the right one.

The results for $conf_6$ show that when action sequence building-block is disabled the intensity of the penalty function could be set even lower. For instance triple_kick is learned faster (9s67 instead of 21s47 for $conf_1$). Conversely the results for $conf_7$ show that when action sequence building-block is enabled, if the Occam's razor is too weak it can dramatically slow down the search. That is because in this circumstance the search is mislead by longer candidates that fit and takes a very cut before it can reach the optimal more compact solution.

32.7 Conclusion

In our experimentation with hillclimbing for learning pet tricks in a virtual world, we have shown that the combination of

1. candidate reduction into normal form,
2. filtering operators to narrow the alphabet,
3. using action sequences that are compatible with the shown behavior as building blocks,
4. adequately choosing and calibrating the complexity penalty function,

can speed up imitation learning so that moderately complex tricks can be learned within seconds to minutes instead of hours, using a simple "hill climbing with restarts" learning algorithm.

While we have discussed these ideas in the context of pet tricks, they have of course been developed with more general applications in mind, and have been applied in many additional contexts. Combo can be used to represent any sort of procedure, and both the hillclimbing algorithm and the optimization heuristics described here appear broad in their relevance.

Natural extensions of the approach described here include the following directions:

1. improving the Entity and Entropy filter using ECAN and PLN so that filtering is not only based on entropy but also relevancy with respect to the context and background knowledge,
2. using transfer learning (see Section 33.4 of Chapter 33) to tune the parameters of algorithm using contextual and background knowledge.

Indeed these improvements are under active investigation at time of writing, and some may well have been implemented and tested by the time you read this.

Chapter 33
Probabilistic Evolutionary Procedure Learning

Co-authored with Moshe Looks*

33.1 Introduction

The CogPrime architecture fundamentally requires, as one of its components, some powerful algorithm for automated program learning. This algorithm must be able to solve procedure learning problems relevant to achieving human-like goals in everyday human environments, relying on the support of other cognitive processes, and providing them with support in turn. The requirement is not that complex human behaviors need to be learnable via program induction alone, but rather than when the best way for the system to achieve a certain goal seems to be the acquisition of a chunk of *procedural* knowledge, the program learning component should be able to carry out the requisite procedural knowledge.

As CogPrime is a fairly broadly-defined architecture overall, there are no extremely precise requirements for its procedure learning component. There could be variants of CogPrime in which procedure learning carried more or less weight, relative to other components.

Some guidance here may be provided by looking at which tasks are generally handled by humans primarily using procedural learning, a topic on which cognitive psychology has a fair amount to say, and which is also relatively amenable to commonsense understanding based on our introspective and social experience of being human. When we know how to do something, but can't explain very clearly to ourselves or others how we do it, the chances are high that we have acquired this knowledge using some form of "procedure learning" as opposed to declarative learning. This is especially the case if we can do this same sort of thing in many different contexts, each time displaying a conceptually similar series of actions, but adapted to the specific situation. We would like CogPrime to be able to carry out procedural learning in roughly the same situations ordinary humans can (and potentially other

* First author

situations as well: maybe even at the start, and definitely as development proceeds), largely via action of its program learning component.

In practical terms, our intuition (based on considerable experience with automated program learning, in OpenCog and other contexts) is that one requires a program learning component capable of learning programs with between dozens and hundreds of program tree nodes, in Combo or some similar representation – not able to learn *arbitrary* programs of this size, but rather able to solve problems arising in everyday human situations in which the simplest acceptable solutions involve programs of this size. We also suggest that the majority of procedure learning problems arising in everyday human situation can be solved via program with hierarchical structure, so that it likely suffices to be able to learn programs with between dozens and hundreds of program tree nodes, where the programs have a modular structure, consisting of modules each possessing no more than dozens of program tree nodes. Roughly speaking, with only a few dozen Combo tree nodes, complex behaviors seem only achievable via using very subtle algorithmic tricks that aren't the sort of thing a human-like mind in the early stages of development could be expected to figure out; whereas, getting beyond a few hundred Combo tree nodes, one seems to get into the domain where an automated program learning approach is likely infeasible without rather strong restrictions on the program structure, so that a more appropriate approach within CogPrime would be to use PLN, concept creation or other methods to fuse together the results of multiple smaller procedure learning runs.

While simple program learning techniques like hillclimbing (as discussed in Chapter 32 above) can be surprisingly powerful, they do have fundamental limitations, and our experience and intuition both indicate that they are not adequate for serving as CogPrime 's primary program learning component. This chapter describes an algorithm that we do believe is thus capable – CogPrime 's most powerful and general procedure learning algorithm, MOSES, an integrative probabilistic evolutionary program learning algorithm that was briefly overviewed in Chapter 6 of Part 1.

While MOSES as currently designed and implemented embodies a number of specific algorithmic and structural choices, at bottom it embodies two fundamental insights that are critical to generally intelligent procedure learning:

- *Evolution is the right approach* to learning of difficult procedures
- *Enhancing evolution with probabilistic methods is necessary.* Pure evolution, in the vein of the evolution of organisms and species, is too slow for broad use within cognition; so what is required is a hybridization of evolutionary and probabilistic methods, where probabilistic methods provide a more directed approach to generating candidate solutions that is possible with typical evolutionary heuristics like crossover and mutation

We summarize these insights in the phrase *Probabilistic Evolutionary Program Learning* (PEPL); MOSES is then one particular PEPL algorithm, and in our view a very good one. We have also considered other related algorithms

such as the PLEASURE algorithm [Goe08a] (which may also be hybridized with MOSES), but for the time being it appears to us that MOSES satisfies CogPrime 's needs.

Our views on the fundamental role of evolutionary dynamics in intelligence were briefly presented in Chapter 3 of Part 1. Terrence Deacon said it even more emphatically: "At every step the design logic of brains is a Darwinian logic: overproduction, variation, competition, selection ... it should not come as a surprise that this same logic is also the basis for the normal millisecond-by-millisecond information processing that continues to adapt neural software to the world." [**?**] He has articulated ways in which, during neurodevelopment, difference computations compete with each other (e.g., to determine which brain regions are responsible for motor control). More generally, he posits a kind of continuous flux as control shifts between competing brain regions, again, based on high-level "cognitive demand."

Deacon's intuition is similar to the one that led Edelman to propose Neural Darwinism [Ede93], and Calvin and Bickerton [CB00] to pose the notion of mind as a "Darwin Machine". The latter have given plausible neural mechanisms ("Darwin Machines") for synthesizing short "programs". These programs are for tasks such as rock throwing and sentence generation, which are represented as coherent firing patterns in the cerebral cortex. A population of such patterns, competing for neurocomputational territory, replicates with variations, under selection pressure to conform to background knowledge and constraints.

To incorporate these insights, a system is needed that can recombine existing solutions in a non-local synthetic fashion, learning nested and sequential structures, and incorporate background knowledge (e.g. previously learned routines). MOSES is a particular kind of *program evolution* intended to satisfy these goals, using a combination of probability theory with ideas drawn from genetic programming, and also incorporating some ideas we have seen in previous chapters such as program normalization.

The main conceptual assumption about CogPrime 's world, implicit in the suggestion of MOSES as the primary program learning component, is that the goal-relevant knowledge that cannot effectively be acquired by the other methods at CogPrime 's disposal (PLN, ECAN, etc.), forms a body of knowledge that can effectively be induced across via probabilistic modeling on the space of programs for controlling a CogPrime agent. If this is not true, then MOSES will provide no advantage over simple methods like well-tuned hillclimbing as described in Chapter 32. If it is true, then the effort of deploying a complicated algorithm like MOSES is worthwhile. In essence, the assumption is that there are relatively simple regularities among the programs implementing those procedures that are most critical for a human-like intelligence to acquire via procedure learning rather than other methods.

33.1.1 Explicit versus Implicit Evolution in CogPrime

Of course, the general importance of evolutionary dynamics for intelligence does not imply the need to use explicit evolutionary algorithms in one's AGI system. Evolution can occur in an intelligent system whether or not the low-level *implementation layer* of the system involves any explicitly evolutionary processes. For instance it's clear that the human mind/brain involves evolution in this sense on the emergent level – we create new ideas and procedures by varying and combining ones that we've found useful in the past, and this occurs on a variety of levels of abstraction in the mind. In CogPrime , however, we have chosen to implement evolutionary dynamics explicitly, as well as encouraging it to occur implicitly.

CogPrime is intended to display evolutionary dynamics on the derived-hypergraph level, and this is intended to be a consequence of both explicitly-evolutionary and not-explicitly-evolutionary dynamics. Cognitive processes such as PLN inference may lead to emergent evolutionary dynamics (as useful logical relationships are reasoned on and combined, leading to new logical relationships in an evolutionary manner); even though PLN in itself is not explicitly *evolutionary* in character, it becomes emergently evolutionary via its coupling with CogPrime 's attention allocation subsystem, which gives more cognitive attention to Atoms with more importance, and hence creates an evolutionary dynamic with importance as the fitness criterion and the whole constellation of MindAgents as the novelty-generation mechanism. However, MOSES *explicitly* embodies evolutionary dynamics for the learning of new patterns and procedures that are too complex for hillclimbing or other simple heuristics to handle. And this evolutionary learning subsystem naturally also contributes to the creation of evolutionary patterns on the emergent, derived-hypergraph level.

33.2 Estimation of Distribution Algorithms

There is a long history in AI of applying evolution-derived methods to practical problem-solving; John Holland's genetic algorithm [Hol75], initially a theoretical model, has been adapted successfully to a wide variety of applications (see e.g. the proceedings of the GECCO conferences). Briefly, the methodology applied is as follows:

1. generate a random population of solutions to a problem
2. evaluate the solutions in the population using a predefined fitness function
3. select solutions from the population proportionate to their fitnesss
4. recombine/mutate them to generate a new population
5. go to step 2

Holland's paradigm has been adapted from the case of fixed-length strings to the evolution of variable-sized and shaped trees (typically Lisp S-expressions), which in principle can represent arbitrary computer programs [Koz92, Koz94].

Recently, replacements-for/extensions-of the genetic algorithm have been developed (for fixed-length strings) which may be described as *estimation-of-distribution* algorithms (see [Pel05] for an overview). These methods, which outperform genetic algorithms and related techniques across a range of problems, maintain centralized probabilistic models of the population learned with sophisticated datamining techniques. One of the most powerful of these methods is the *Bayesian optimization algorithm* (BOA) [Pel05].

The basic steps of the BOA are:

1. generate a random population of solutions to a problem
2. evaluate the solutions in the population using a predefined fitness function
3. from the promising solutions in the population, learn a generative model
4. create new solutions using the model, and merge them into the existing population
5. go to step 2.

The neurological implausibility of this sort of algorithm is readily apparent – yet recall that in CogPrime we are attempting to roughly emulate human cognition on the level of behavior not structure or dynamics.

Fundamentally, the BOA and its ilk (the *competent* adaptive optimization algorithms) differ from classic selecto-recombinative search by attempting to dynamically learn a problem decomposition, in terms of the variables that have been pre-specified. The BOA represents this decomposition as a Bayesian network (directed acyclic graph with the variables as nodes, and an edge from x to y indicating that y is probabilistically dependent on x). An extension, the hierarchical Bayesian optimization algorithm (hBOA) , uses a Bayesian network with local structure to more accurately represent hierarchical dependency relationships. The BOA and hBOA are scalable and robust to noise across the range of nearly decomposable functions. They are also effective, empirically, on real-world problems with unknown decompositions, which may or may not be effectively representable by the algorithms; robust, high-quality results have been obtained for Ising spin glasses and MaxSAT, as well as a real-world telecommunication problem (see [**?**] for references).

33.3 Competent Program Evolution via MOSES

In this section we summarize *meta-optimizing semantic evolutionary search* (MOSES), a system for competent program evolution, described more thoroughly in [Loo06]. Based on the viewpoint developed in the previous section, MOSES is designed around the central and unprecedented capability of competent optimization algorithms such as the hBOA, to generate new solutions

that simultaneously combine sets of promising assignments from previous solutions according to a dynamically learned problem decomposition. The novel aspects of MOSES described herein are built around this core to exploit the unique properties of program learning problems. This facilitates effective problem decomposition (and thus competent optimization).

33.3.1 Statics

The basic goal of MOSES is to exploit the regularities in program spaces outlined in the previous section, most critically *behavioral decomposability* and *white box execution*, to dynamically construct representations that limit and transform the program space being searched into a relevant subspace with a compact problem decomposition. These representations will evolve as the search progresses.

33.3.1.1 An Example

Let's start with an easy example. What knobs (meaningful parameters to vary) exist for the family of programs depicted in Figure **??** on the left? We can assume, in accordance with the principle of white box execution, that all symbols have their standard mathematical interpretations, and that $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ are real-valued variables.

In this case, all three programs correspond to variations on the behavior represented graphically on the right in the figure. Based on the principle of behavioral decomposability, good knobs should express plausible evolutionary variation and recombination of features in behavior space, regardless of the nature of the corresponding changes in program space. It's worth repeating once more that this goal cannot be meaningfully addressed on a syntactic level - it requires us to leverage background knowledge of what the symbols in our vocabulary (***cos***, $+$, ***0.35***, etc.) actually *mean*.

A good set of knobs will also be *orthogonal*. Since we are searching through the space of combinations of knob settings (not a single change at a time, but a set of changes), any knob whose effects are equivalent to another knob or combination of knobs is undesirable.† Correspondingly, our set of knobs should *span* all of the given programs (i.e., be able to represent them as various knob settings).

A small *basis* for these programs could be the 3-dimensional parameter space, $x_1 \in \{x, z, 0\}$ (left argument of the root node), $x_2 \in \{y, x\}$ (argument of ***cos***), and $x_3 \in [0.3, 0.4]$ (multiplier for the ***cos***-expression). However, this

† First because this will increase the number of samples needed to effectively model the structure of knob-space, and second because this modeling will typically be quadratic with the number of knobs, at least for the BOA or hBOA.

is a very limiting view, and overly tied to the particulars of how these three programs happen to be encoded. Considering the space *behaviorally* (right of Figure **??**), a number of additional knobs can be imagined which might be turned in meaningful ways, such as:

1. numerical constants modifying the phase and frequency of the cosine expression,
2. considering some weighted average of $\boldsymbol{x}$ and $\boldsymbol{y}$ instead of one or the other,
3. multiplying the entire expression by a constant,
4. adjusting the relative weightings of the two arguments to $+$.

33.3.1.2 Syntax and Semantics

This kind of representation-building calls for a correspondence between syntactic and semantic variation. The properties of program spaces that make this difficult are over-representation and chaotic execution, which lead to *non-orthogonality*, *oversampling of distant behaviors*, and *undersampling of nearby behaviors*, all of which can directly impede effective program evolution.

Non-orthogonality is caused by over-representation. For example, based on the properties of commutativity and associativity, $a_1 + a_2 + ... + a_n$ may be expressed in exponentially many different ways, if $+$ is treated as a non-commutative and non-associative binary operator. Similarly, operations such as addition of zero and multiplication by one have no effect, the successive addition of two constants is equivalent to the addition of their sum, etc. These effects are not quirks of real-valued expressions; similar redundancies appear in Boolean formulae ($\boldsymbol{x}$ ***AND*** $\boldsymbol{x} \equiv \boldsymbol{x}$), list manipulation (***cdr(cons(x, L))*** $\equiv \boldsymbol{L}$), and conditionals (***if x then y else z*** $\equiv$ ***if NOT x then z else y***).

Without the ability to exploit these identities, we are forced to work in a greatly expanded space which represents equivalent expression in many different ways, and will therefore be very far from orthogonality. Completely eliminating redundancy is infeasible, and typically NP-hard (in the domain of Boolean formulae it is reducible to the satisfiability problem, for instance), but one can go quite far with a heuristic approach.

Oversampling of distant behaviors is caused directly by chaotic execution, as well as a somewhat subtle effect of over-representation, which can lead to simpler programs being heavily oversampled. Simplicity is defined relative to a given program space in terms of minimal length, the number of symbols in the shortest program that produces the same behavior.

Undersampling of nearby behaviors is the flip side of the oversampling of distant behaviors. As we have seen, syntactically diverse programs can have the same behavior; this can be attributed to redundancy, as well as non-redundant programs that simply compute the same result by different means. For example, ***3*x*** can also be computed as $\boldsymbol{x}+\boldsymbol{x}+\boldsymbol{x}$; the first version uses less symbols, but neither contains any obvious "bloat" such as addition

of zero or multiplication by one. Note however that the nearby behavior of ***3.1*x***, is syntactically close to the former, and relatively far from the latter. The converse is the case for the behavior of ***2*x+y***. In a sense, these two expressions can be said to exemplify differing organizational principles, or points of view, on the underlying function.

Differing organizational principles lead to different biases in sampling nearby behaviors. A superior organizational principle (one leading to higher-fitness syntactically nearby programs for a particular problem) might be considered a *metaptation* (adaptation at the second tier), in the terminology of King [21]. Since equivalent programs organized according to different principles will have identical fitnesss, some methodology beyond selection for high fitnesss must be employed to search for good organizational principles. Thus, the resolution of undersampling of nearby behaviors revolves around the management of *neutrality* in search, a complex topic beyond the scope of this chapter.

These three properties of program spaces greatly affect the performance of evolutionary methods based solely on syntactic variation and recombination operators, such as local search or genetic programming. In fact, when quantified in terms of various fitness-distance correlation measures, they can be effective predictors of algorithm performance, although they are of course not the whole story. A semantic search procedure will address these concerns in terms of the underlying behavioral effects of and interactions between a language's basic operators; the general scheme for doing so in MOSES is the topic of the next subsection.

33.3.1.3 Neighborhoods and Normal Forms

The procedure MOSES uses to construct a set of knobs for a given program (or family of structurally related programs) is based on three conceptual steps: *reduction to normal form*, *neighborhood enumeration*, and *neighborhood reduction*.

Reduction to normal form

- in this step, redundancy is heuristically eliminated by reducing programs to a *normal form*. Typically, this will be via the iterative application of a series of local rewrite rules (e.g., $\forall x,\ x+0 \rightarrow x$), until the target program no longer changes. Note that the well-known conjunctive and disjunctive normal forms for Boolean formulae are generally unsuitable for this purpose; they destroy the hierarchical structure of formulae, and dramatically limit the range of behaviors (in this case Boolean functions) that can be expressed compactly. Rather, *hierarchical normal forms* for programs are required.

Neighborhood enumeration

- in this step, a set of possible atomic *perturbations* is generated for all programs under consideration (the overall perturbation set will be the union of these). The goal is to heuristically generate new programs that correspond to behaviorally nearby variations on the source program, in such a way that arbitrary sets of perturbations may be *composed* combinatorially to generate novel valid programs.

Neighborhood reduction

- in this step, redundant perturbations are heuristically culled to reach a more orthogonal set. A straightforward way to do this is to exploit the reduction to normal form outlined above; if multiple knobs lead to the same normal forms program, only one of them is actually needed. Additionally, note that the number of symbols in the normal form of a program can be used as a heuristic approximation for its minimal length - if the reduction to normal form of the program resulting from twiddling some knob significantly decreases its size, it can be assumed to be a source of oversampling, and hence eliminated from consideration. A slightly smaller program is typically a meaningful change to make, but a large reduction in complexity will rarely be useful (and if so, can be accomplished through a combination of knobs that individually produce small changes).

At the end of this process, we will be left with a set of knobs defining a subspace of programs centered around a particular region in program space and heuristically centered around the corresponding region in behavior space as well. This is part of the *meta* aspect of MOSES, which seeks not to evaluate variations on existing programs itself, but to construct parameterized program subspaces (representations) containing meaningful variations, guided by background knowledge. These representations are used as search spaces within which an optimization algorithm can be applied.

33.3.2 Dynamics

As described above, the representation-building component of MOSES constructs a parameterized representation of a particular *region* of program space, centered around a single program of family of closely related programs. This is consonant with the line of thought developed above, that a representation constructed across an arbitrary region of program space (e.g., all programs containing less than $\boldsymbol{n}$ symbols), or spanning an arbitrary collection of unrelated programs, is unlikely to produce a meaningful parameterization (i.e., one leading to a compact problem decomposition).

A sample of programs within a region derived from representation-building together with the corresponding set of knobs will be referred to herein as a *deme*;[‡] a set of demes (together spanning an arbitrary area within program space in a patchwork fashion) will be referred to as a *metapopulation*.[§] MOSES operates on a metapopulation, adaptively creating, removing, and allocating optimization effort to various demes. Deme management is the second fundamental *meta* aspect of MOSES, after (and above) representation-building; it essentially corresponds to the problem of effectively allocating computational resources to competing regions, and hence to competing programmatic organizational- representational schemes.

33.3.2.1 Algorithmic Sketch

The salient aspects of programs and program learning lead to requirements for competent program evolution that can be addressed via a representation-building process such as the one shown above, combined with effective deme management. The following sketch of MOSES, elaborating Figure 33.1 repeated here from Chapter 8 of Part 1, presents a simple control flow that dynamically integrates these processes into an overall program evolution procedure:

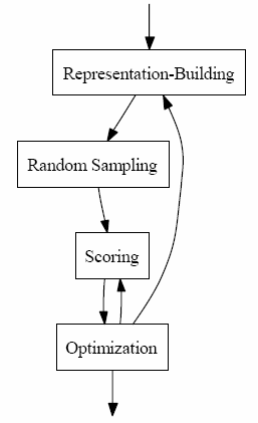


Fig. 33.1 The top-level architectural components of MOSES, with directed edges indicating the flow of information and program control.

[‡] A term borrowed from biology, referring to a somewhat isolated local population of a species.

[§] Another term borrowed from biology, referring to a group of somewhat separate populations (the demes) that nonetheless interact.

1. Construct an initial set of knobs based on some prior (e.g., based on an empty program) and use it to generate an initial random sampling of programs. Add this deme to the metapopulation.
2. Select a deme from the metapopulation and update its sample, as follows:
   a. Select some promising programs from the deme's existing sample to use for modeling, according to the fitness function.
   b. Considering the promising programs as collections of knob settings, generate new collections of knob settings by applying some (competent) optimization algorithm.
   c. Convert the new collections of knob settings into their corresponding programs, reduce the programs to normal form, evaluate their fitnesss, and integrate them into the deme's sample, replacing less promising programs.
3. For each new program that meet the criterion for creating a new deme, if any:
   a. Construct a new set of knobs (via representation-building) to define a region centered around the program (the deme's *exemplar)*, and use it to generate a *new* random sampling of programs, producing a new deme.
   b. Integrate the new deme into the metapopulation, possibly displacing less promising demes.
4. Repeat from step 2.

The criterion for creating a new deme is *behavioral non-dominance* (programs which are not dominated by the exemplars of any existing demes are used as exemplars to create new demes), which can be defined in a domain-specific fashion. As a default, the fitness function may be used to induce dominance, in which case the set of exemplar programs for demes corresponds to the set of top-fitness programs.

33.3.3 Architecture

The preceding algorithmic sketch of MOSES leads to the top-level architecture depicted in Figure **??**. Of the four top-level components, only the fitness function is problem-specific. The representation-building process is domain-specific, while the random sampling methodology and optimization algorithm are domain-general. There is of course the possibility of improving performance by incorporating domain and/or problem-specific bias into random sampling and optimization as well.

33.3.4 Example: Artificial Ant Problem

Let's go through all of the steps that are needed to apply MOSES to a small problem, the artificial ant on the Santa Fe trail [Koz92], and describe the search process. The artificial ant domain is a two-dimensional grid landscape where each cell may or may not contain a piece of food. The artificial ant has a location (a cell) and orientation (facing up, down, left, or right), and navigates the landscape via a primitive sensor, which detects whether or not there is food in the cell that the ant is facing, and primitive actuators *move* (take a single step forward), *right* (rotate 90 degrees clockwise), and *left* (rotate 90 degrees counter-clockwise). The Santa Fe trail problem is a particular 32 x 32 toroidal grid with food scattered on it (Figure 4), and a fitness function counting the number of unique pieces of food the ant eats (by entering the cell containing the food) within 600 steps (movement and 90 degree rotations are considered single steps).

Programs are composed of the primitive actions taking no arguments, a conditional (*if-food-ahead*),¶ which takes two arguments and evaluates one or the other based on whether or not there is food ahead, and *progn*, which takes a variable number of arguments and sequentially evaluates all of them from left to right. To fitness a program, it is evaluated continuously until 600 time steps have passed, or all of the food is eaten (whichever comes first). Thus for example, the program *if-food-ahead(m, r)* moves forward as long as there is food ahead of it, at which point it rotates clockwise until food is again spotted. It's can successfully navigate the first two turns of the placeSanta Fe trail, but cannot cross "gaps" in the trail, giving it a final fitness of 11.

The first step in applying MOSES is to decide what our reduction rules should look like. This program space has several clear sources of redundancy leading to over-representation that we can eliminate, leading to the following reduction rules:

1. Any sequence of rotations may be reduced to either a left rotation, a right rotation, or a reversal, for example:

   progn(left, left, left)
   reduces to
   right

1. Any *if-food-ahead* statement which is the child of an *if-food-ahead* statement may be eliminated, as one of its branches is clearly irrelevant, for example:

   if-food-ahead(m, if-food-ahead(l, r))
   reduces to

¶ This formulation is equivalent to using a general three-argument if-then-else statement with a predicate as the first argument, as there is only a single predicate (food-ahead) for the ant problem.

if-food-ahead(m, r)

1. Any *progn* statement which is the child of a *progn* statement may be eliminated and replaced by its children, for example:

progn(progn(left, move), move)
reduces to
progn(left, move, move)

The representation language for the ant problem is simple enough that these are the only three rules needed – in principle there could be many more. The first rule may be seen as a consequence of general domain-knowledge pertaining to rotation. The second and third rules are fully general simplification rules based on the semantics of *if-then-else* statements and associative functions (such as *progn*), respectively.

These rules allow us to naturally parameterize a knob space corresponding to a given program (note that the arguments to the *progn* and *if-food-ahead* functions will be recursively reduced and parameterized according to the same procedure). Rotations will correspond to knobs with four possibilities (*left, right, reversal, no rotation).* Movement commands will correspond to knobs with two possibilities (*move, no movement).* There is also the possibility of introducing a new command in between, before, or after, existing commands. Some convention (a "canonical form") for our space is needed to determine how the knobs for new commands will be introduced. A representation consists of a rotation knob, followed by a conditional knob, followed by a movement knob, followed by a rotation knob, etc.[‖]

The structure of the space (how large and what shape) and default knob values will be determined by the "exemplar" program used to construct it. The default values are used to bias the initial sampling to focus around the prototype associated to the exemplar: all of the n direct neighbors of the prototype are first added to the sample, followed by a random selection of n programs at a distance of two from the prototype, n programs at a distance of three, etc., until the entire sample is filled. Note that the hBOA can of course effectively recombine this sample to generate novel programs at any distance from the initial prototype. The empty program *progn* (which can be used as the initial exemplar for MOSES), for example, leads to the following prototype:

```
progn(
        rotate? [default no rotation],
        if-food-ahead(
                        progn(
                                rotate? [default no rotation],
```

[‖] That there be some fixed ordering on the knobs is important, so that two rotation knobs are not placed next to each other (as this would introduce redundancy). In this case, the precise ordering chosen (rotation, conditional, movement) does not appear to be critical.

move? [default no movement]),
progn(
rotate? [default no rotation],
move? [default no movement])),
move? [default no movement])

There are six parameters here, three which are quaternary (*rotate*), and three which are binary (*move*). So the program

progn(left, if-food-ahead(move, left))

would be encoded in the space as

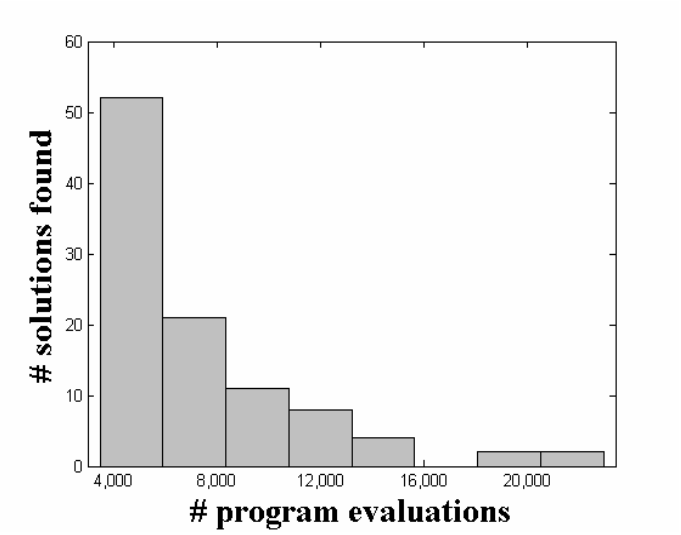
[*left, no rotation, move, left, no movement, no movement*]

with knobs ordered according to a pre-order left-to-right traversal of the program's parse tree (this is merely for exposition; the ordering of the parameters has no effect on MOSES). For a prototype program already containing an *if-food-ahead* statement, nested conditionals would be considered.

A space with six parameters in it is small enough that MOSES can reliably find the optimum (the program *progn(right, if-food-ahead(progn(),left), move)*), with a very small population. After no further improvements have been made in the search for a specified number of generations (calculated based on the size of the space based on a model derived from [23] that is general to the hBOA, and not at all tuned for the artificial ant problem), a new representation is constructed centered around this program.** Additional knobs are introduced "in between" all existing ones (e.g., an optional move in between the first rotation and the first conditional), and possible nested conditionals are considered (a nested conditional occurring in a sequence *after* some other action has been taken is not redundant). The resulting space has 39 knobs, still quite tractable for hBOA, which typically finds a global optimum within a few generations. If the optimum were not to be found, MOSES would construct a new (possibly larger or smaller) representation, centered around the best program that *was* found, and the process would repeat.

The artificial ant problem is well-studied, with published benchmark results available for genetic programming as well as evolutionary programming based solely on mutation (i.e., a form of population-based stochastic hill climbing). Furthermore, an extensive analysis of the search space has been carried out by Langdon and Poli [LP02], with the authors concluding:

** MOSES reduces the exemplar program to normal form before constructing the representation; in this particular case however, no transformations are needed. Similarly, in general neighborhood reduction would be used to eliminate any extraneous knobs (based on domain-specific heuristics). For the ant domain however no such reductions are necessary.



Technique	Computational Effort
Genetic Programming [10]	450,000 evaluations
Evolutionary Programming [24]	136,000 evaluations
MOSES	23,000 evaluations

Fig. 33.2 On the top, histogram of the number of global optima found after a given number of program evaluations for 100 runs of MOSES on the artificial ant problem (each run is counted once for the first global optimum reached). On the bottom, computational effort required to find an optimal solution for various techniques with probability p=.99 (for MOSES p=1, since an optimal solution was found in all runs).

1. The problem is "deceptive at all levels", meaning that the partial solutions that must be recombined to solve the problem to optimality have lower average fitness than the partial solutions that lead to inferior local optima.
2. The search space contains many symmetries (e.g., between left and right rotations),
3. There is an unusually high density of global optima in the space (relative to other common test problems);
4. even though current evolutionary methods can solve the problem, they are not significantly more effective (in terms of the number of program evaluations require) than random sample.
5. "If real program spaces have the above characteristics (we expect them to do so but be still worse) then it is important to be able to demonstrate scalable techniques on such problem spaces".

33.3.4.1 Test Results

Koza [Koz92] reports on a set of 148 runs of genetic programming with a population size of 500 which had a 16% success rate after 51 generations when the runs were terminated (a total of 25,500 program evaluations per run). The minimal "computational effort" needed to achieve success with 99% probability was attained by processing through generation 14 was 450,000 (based on parallel independent runs). Chellapilla [Che97] reports 47 out of 50 successful runs with a minimal computational effort (again, for success with 99% probability) of 136,000 for his stochastic hill climbing method.

In our experiment with the artificial ant problem, one hundred runs of MOSES were executed. Beyond the domain knowledge embodied in the reduction and knob construction procedure, the only parameter that needed to be set was the population scaling factor, which was set to 30 (MOSES automatically adjusts to generate a larger population as the size of the representation grows, with the base case determined by this factor). Based on these "factory" settings, MOSES found optimal solutions on every run out of 100 trials, within a maximum of 23,000 program evaluations (the computational effort figure corresponding to 100% success). The average number of program evaluation required was 6952, with 95% confidence intervals of ±856 evaluations.

Why does MOSES outperform other techniques? One factor to consider first is that the language programs are evolved in is slightly more expressive than that used for the other techniques; specifically, a *progn* is allowed to have no children (if all of its possible children are "turned off"), leading to the possibility of *if-food-ahead* statements which do nothing if food is present (or not present). Indeed, many of the smallest solutions found by MOSES exploit this feature. This can be tested by inserting a "do nothing" operation into the terminal set for genetic programming (for example). Indeed, this reduces the computational effort to 272,000; an interesting effect, but still over an order of magnitude short of the results obtained with MOSES (the success rate after 50 generations is still only 20%).

Another possibility is that the reductions in the search space via simplification of programs alone are responsible. However, the results past attempts at introducing program simplification into genetic programming systems [27, 28] have been mixed; although the system may be sped up (because programs are smaller), there have been no dramatic improvement in results noted. To be fair, these results have been primarily focused on the symbolic regression domain; I am not aware of any results for the artificial ant problem.

The final contributor to consider is the sampling mechanism (knowledge-driven knob-creation followed by probabilistic model-building). We can test to what extent model-building contributes to the bottom line by simply disabling it and assuming probabilistic independence between all knobs. The result here is of interest because model-building can be quite expensive ($O(n2N)$ per generation, where n is the problem size and N is the popu-

lation size[††]). In 50 independent runs of MOSES without model-building, a global optimum was still discovered in all runs. However, the variance in the number of evaluations required was much higher (in two cases over 100,000 evaluations were needed). The new average was 26,355 evaluations to reach an optimum (about 3.5 times more than required with model-building). The contribution of model-building to the performance of MOSES is expected to be even greater for more difficult problems.

Applying MOSES without model-building (i.e., a model assuming no interactions between variables) is a way to test the combination of representation-building with an approach resembling the probabilistic incremental program learning (PIPE) algorithm [SS03a], which learns programs based on a probabilistic model without any interactions. PIPE has now been shown to provide results competitive with genetic programming on a number of problems (regression, agent control, etc.).

It is additionally possible to look inside the models that the hBOA constructs (based on the empirical statistics of successful programs) to see what sorts of linkages between knobs are being learned.[‡‡] For the 6-knob model given above for instance an analysis the linkages learned shows that the three most common pairwise dependencies uncovered, occurring in over 90% of the models across 100 runs, are between the rotation knobs. No other individual dependencies occurred in more than 32% of the models. This preliminary finding is quite significant given Landgon and Poli's findings on symmetry, and their observation that "[t]hese symmetries lead to essentially the same solutions appearing to be the opposite of each other. E.g. either a pair of Right or pair of Left terminals at a particular location may be important."

In this relatively simple case, all of the components of MOSES appear to mesh together to provide superior performance – which is promising, though it of course does not prove that these same advantages will apply across the range of problems relevant to human-level AGI.

33.3.5 Discussion

The overall MOSES design is unique. However, it is instructive at this point to compare its two primary unique facets (representation-building and deme management) to related work in evolutionary computation.

Rosca's *adaptive representation* architecture [Ros99] is an approach to program evolution which also alternates between separate representation-building and optimization stages. It is based on Koza's genetic programming, and modifies the representation based on a *syntactic* analysis driven by the

[††] The fact that reduction to normal tends to reduce the problem size is another synergy between it and the application of probabilistic model-building.

[‡‡] There is in fact even more information available in the hBOA models concerning hierarchy and direction of dependence, but this is difficult to analyze.

fitness function, as well as a modularity bias. The representation-building that takes place consists of introducing new compound operators, and hence modifying the implicit distance function in tree-space. This modification is uniform, in the sense that the new operators can be placed in any context, without regard for semantics.

In contrast to Rosca's work and other approaches to representation-building such as Koza's *automatically defined functions* [KA95], MOSES explicitly addresses on the underlying (semantic) structure of program space independently of the search for any kind of modularity or problem decomposition. This preliminary stage critically changes neighborhood structures (syntactic similarity) and other aggregate properties of programs.

Regarding deme management, the embedding of an evolutionary algorithm within a superordinate procedure maintaining a metapopulation is most commonly associated with "island model" architectures [**?**]. One of the motivations articulated for using island models has been to allow distinct islands to (usually implicitly) explore different regions of the search space, as MOSES does explicitly. MOSES can thus be seen as a very particular kind of island model architecture, where programs never migrate between islands (demes), and islands are created and destroyed dynamically as the search progresses.

In MOSES, optimization does not operate directly on program space, but rather on a subspace defined by the representation-building process. This subspace may be considered as being defined by a sort of template assigning values to some of the underlying dimensions (e.g., it restricts the size and shape of any resulting trees). The messy genetic algorithm [GKD89], an early competent optimization algorithm, uses a similar mechanism - a common "competitive template" is used to evaluate candidate solutions to the optimization problem which are themselves underspecified. Search consequently centers on the template(s), much as search in MOSES centers on the programs used to create new demes (and thereby new representations). The issue of deme management can thus be seen as analogous to the issue of template selection in the messy genetic algorithm.

33.3.6 Conclusion

Competent evolutionary optimization algorithms are a pivotal development, allowing encoded problems with compact decompositions to be tractably solved according to normative principles. We are still faced with the problem of *representation-building* – casting a problem in terms of knobs that can be twiddled to solve it. Hopefully, the chosen encoding will allow for a compact problem decomposition. Program learning problems in particular rarely possess compact decompositions, due to particular features generally present in program spaces (and in the mapping between programs and behaviors). This often leads to intractable problem formulations, even if the

mapping between behaviors and fitness has an intrinsic separable or nearly decomposable structure. As a consequence, practitioners must often resort to manually carrying out the analogue of representation-building, on a problem-specific basis. Working under the thesis that *the properties of programs and program spaces can be leveraged as inductive bias to remove the burden of manual representation-building, leading to competent program evolution*, we have developed the MOSES system, and explored its properties.

While the discussion above has highlighted many of the features that make MOSES uniquely powerful, in a sense it has told only half the story. Part of what makes MOSES valuable for CogPrime is that it's good on its own; and the other part is that it cooperates well with the other cognitive processes within CogPrime . We have discussed aspects of this already in Chapter 8 of Part 1, especially in regard to the MOSES/PLN relationship. In the following section we proceed further to explore the interaction of MOSES with other aspects of the CogPrime system – a topic that will arise repeatedly in later chapters as well.

33.4 Supplying Evolutionary Learning with Long-Term Memory

This section introduces an important enhancement to evolutionary learning, which extends the basic PEPL framework, by forming an adaptive hybridization of PEPL optimization with PLN inference (rather than merely using PLN inference within evolutionary learning to aid with modeling).

The first idea here is the use of PLN to supply evolutionary learning with a long-term memory. Evolutionary learning approaches each problem as an isolated entity, but in reality, an CogPrime system will be confronting a long series of optimization problems, with subtle interrelationships. When trying to optimize the function f, CogPrime may make use of its experience in optimizing other functions g.

Inference allows optimizers of g to be analogically transformed into optimizers of f, for instance it allows one to conclude:

```
Inheritance f g
   EvaluationLink f x
   EvaluationLink g x
```

However, less obviously, inference also allows patterns in populations of optimizers of g to be analogically transformed into patterns in populations of optimizers of f. For example, if *pat* is a pattern in good optimizers of f, then we have:

```
InheritanceLink f g

ImplicationLink
   EvaluationLink f x
```

```
   EvaluationLink pat x

ImplicationLink
   EvaluationLink g x
   EvaluationLink pat x
```

(with appropriate probabilistic truth values), an inference which says that patterns in the population of f-optimizers should also be patterns in the population of g-optimizers).

Note that we can write the previous example more briefly as:

```
InheritanceLink f g
   ImplicationLink (EvaluationLink f) (EvaluationLink pat)
   ImplicationLink (EvaluationLink g) (EvaluationLink pat)
```

A similar formula holds for SimilarityLinks.

We may also infer:

```
ImplicationLink (EvaluationLink g) (EvaluationLink pat_g)

ImplicationLink (EvaluationLink f) (EvaluationLink pat_f)

ImplicationLink
    (EvaluationLink (g AND f))
    (EvaluationLink (pat_g AND pat_f))
```

and:

```
ImplicationLink (EvaluationLink f) (EvaluationLink pat)

ImplicationLink (EvaluationLink ~f) (EvaluationLink ~pat)
```

Through these sorts of inferences, PLN inference can be used to give evolutionary learning a long-term memory, allowing knowledge about population models to be transferred from one optimization problem to another. This complements the more obvious use of inference to transfer knowledge about specific solutions from one optimization problem to another.

For instance in the problem of finding a compact program generating some given sequences of bits the system might have noticed that when the number of 0 roughly balances the number of 1 (let us call this property STR_BALANCE) successful optimizers tend to give greater biases toward conditionals involving comparisons of the number of 0 and 1 inside the condition, let us call this property over optimizers COMP_CARD_DIGIT_BIAS. This can be expressed in PLN as follows

```
AverageQuantifierLink ⟨tv⟩
    ListLink
        $X
        $Y
    ImplicationLink
        ANDLink
            InheritanceLink
                STR_BALANCE
                $X
            EvaluationLink
                SUCCESSFUL_OPTIMIZER_OF
                ListLink
                    $Y
                    $X
        InheritanceLink
            COMP_CARD_DIGIT_BIAS
            $Y
```

which translates by, if the problem $X inherits from STR_BALANCE and $Y is a successful optimizer of $X then, with probability p calculated according to tv, $Y tends to be biased according to the property described by COMP_CARD_DIGIT_BIAS.

33.5 Hierarchical Program Learning

Next we discuss hierarchical program structure, and its reflection in probabilistic modeling, in more depth. This is a surprisingly subtle and critical topic, which may be approached from several different complementary angles. To an extent, hierarchical structure is automatically accounted for in MOSES, but it may also be valuable to pay more explicit mind to it.

In human-created software projects, one common approach for dealing with the existence of complex interdependencies between parts of a program is to give the program a hierarchical structure. The program is then a hierarchical arrangement of programs within programs within programs, each one of which has relatively simple dependencies between its parts (however its parts may themselves be hierarchical composites). This notion of hierarchy is essential to such programming methodologies as modular programming and object-oriented design.

Pelikan and Goldberg discuss the hierarchal nature of human problem-solving, in the context of the hBOA (hierarchical BOA) version of BOA. However, the hBOA algorithm does not incorporate hierarchical program structure nearly as deeply and thoroughly as the hierarchical procedure learning approach proposed here. In hBOA the hierarchy is implicit in the models of the evolving population, but the population instances themselves are not

necessarily explicitly hierarchical in structure. In hierarchical PEPL as we describe it here, the population consists of hierarchically structured Combo trees, and the hierarchy of the probabilistic models corresponds directly to this hierarchical program structure.

The ideas presented here have some commonalities to John Koza's ADFs and related tricks for putting reusable subroutines in GP trees, but there are also some very substantial differences, which we believe will make the current approach far more effective (though also involving considerably more computational overhead).

We believe that this sort of hierarchically-savvy modeling is what will be needed to get probabilistic evolutionary learning to scale to large and complex programs, just as hierarchy-based methodologies like modular and object-oriented programming are needed to get human software engineering to scale to large and complex programs.

33.5.1 Hierarchical Modeling of Composite Procedures in the AtomSpace

The possibility of hierarchically structured programs is (intentionally) present in the CogPrime design, even without any special effort to build hierarchy into the PEPL framework. Combo trees may contain Nodes that point to PredicateNodes, which may in turn contain Combo trees, etc. However, our current framework for learning Combo trees does not take advantage of this hierarchy. What is needed, in order to do so, is for the models used for instance generation to include events of the form:

Combo tree Node at position x has type PredicateNode; and the PredicateNode at position x contains a Combo tree that possesses property P.

where x is a position in a Combo tree and P is a property that may or may not be true of any given Combo tree. Using events like this, a relatively small program explicitly incorporating only *short-range dependencies*; may implicitly encapsulate long-range dependencies via the properties P.

But where do these properties P come from? These properties should be patterns learned as part of the probabilistic modeling of the Combo tree inside the PredicateNode at position x. For example, if one is using a decision tree modeling framework, then the properties might be of the form *decision tree D evaluates to True*. Note that not all of these properties have to be statistically correlated with the fitness of the PredicateNode at position x (although some of them surely will be).

Thus we have a multi-level probabilistic modeling strategy. The top-level Combo tree has a probabilistic model whose events may refer to patterns that are parts of the probabilistic models of Combo trees that occur within it and so on down.

In instance generation, when a newly generated Combo tree is given a PredicateNode at position x, two possibilities exist:

- There is already a model for PredicateNodes at position x in Combo trees in the given population, in which case a population of PredicateNodes potentially living at that position are drawn from the known model, and evaluated.
- There is no such model (because it has never been tried to create a PredicateNode at position x in this population before), in which case a new population of Combo trees is created corresponding to the position, and evaluated.

Note that the fitness of a Combo tree that is not at the top level of the overall process, is assessed indirectly in terms of the fitness of the higher-level Combo tree in which it is embedded, due to the requirement of having certain properties, etc.

Suppose each Combo tree in the hierarchy has on average R adaptable sub-programs (represented as Nodes pointing to PredicateNodes containing Combo trees to be learned). Suppose the hierarchy is K levels deep. Then we will have about $R \times K$ program tree populations in the tree. This suggests that hierarchies shouldn't get too big, and indeed, they shouldn't need to, for the same essential reason that human-created software programs, if well-designed, tend not to require extremely deep and complex hierarchical structures.

One may also introduce a notion of *reusable components* across various program learning runs, or across several portions of the same hierarchical program. Here is one learning patterns of the form:

If property $P1(C, x)$ applies to a Combo tree C and a node x within it, then it is often good for node x to refer to a PredicateNode containing a Combo tree with property $P2$.

These patterns may be assigned probabilities and may be used in instance generation. They are general or specialized *programming guidelines,* which may be learned over time.

33.5.2 Identifying Hierarchical Structure In Combo trees via MetaNodes and Dimensional Embedding

One may also apply the concepts of the previous section to model a population of CTs that doesn't explicitly have an hierarchical structure, via introducing the hierarchical structure during the evolutionary process, through the introduction of special extra Combo tree nodes called MetaNodes. This may be done in a couple different ways, here we will introduce a simple way of doing this based on dimensional embedding, and then in the next section we will allude to a more sophisticated approach that uses inference instead.

The basic idea is to couple decision tree modeling with dimensional embedding of subtrees, a trick that enables small decision tree models to cover large regions of a CT in an approximate way, and which leads naturally to a form of probabilistically-guided crossover.

The approach as described here works most simply for CT's that have many subtrees that can be viewed as mapping numerical inputs into numerical outputs. There are clear generalizations to other sorts of CT's, but it seems advisable to test the approach on this relatively simple case first.

The first part of the idea is to represent subtrees of a CT as numerical vectors in a relatively low-dimensional space (say N=50 dimensions). This can be done using our existing dimensional embedding algorithm, which maps any metric space of entities into a dimensional space. All that's required is that we define a way of measuring distance between subtrees. If we look at subtrees with numerical inputs and outputs, this is easy. Such a subtree can be viewed as a function mapping Rn into Rm; and there are many standard ways to calculate the distance between two functions of this sort (for instance one can make a Monte Carlo estimate of the L^p metric which is defined as:

```
[Sum{x} (f(x) - g(x))^p] ^ (1/p)
```

Of course, the same idea works for subtrees with non-numerical inputs and outputs, the tuning and implementation are just a little trickier.

Next, one can augment a CT with meta-nodes that correspond to subtrees. Each meta-node is of a special CT node type MetaNode, and comes tagged with an N-dimensional vector. Exactly which subtrees to replace with MetaNodes is an interesting question that must be solved via some heuristics.

Then, in the course of executing the PEPL algorithm, one does decision tree modeling as usual, but making use of MetaNodes as well as ordinary CT nodes. The modeling of MetaNodes is quite similar to the modeling of Nodes representing ConceptNodes and PredicateNodes using embedding vectors. In this way, one can use standard, small decision tree models to model fairly large portions of CT's (because portions of CT's are approximately represented by MetaNodes).

But how does one do instance generation, in this scheme? What happens when one tries to do instance generation using a model that predicts a MetaNode existing in a certain location in a CT? Then, the instance generation process has got to find some CT subtree to put in the place where the MetaNode is predicted. It needs to find a subtree whose corresponding embedding vector is close to the embedding vector stored in the MetaNode. But how can it find such a subtree?

There seem to be two ways:

1. A reasonable solution is to look at the database of subtrees that have been seen before in the evolving population, and choose one from this database, with the probability of choosing subtree X being proportional to the distance between X's embedding vector and the embedding vector stored in the MetaNode.

2. One can simply choose good subtrees, where the goodness of a subtree is judged by the average fitness of the instances containing the target subtree.

One can use a combination of both of these processes during instance generation.

But of course, what this means is that we're in a sense doing a form of crossover, because we're generating new instances that combine subtrees from previous instances. But we're combining subtrees in a judicious way guided by probabilistic modeling, rather than in a random way as in GP-style crossover.

33.5.2.1 Inferential MetaNodes

MetaNodes are an interesting and potentially powerful technique, but we don't believe that they, or any other algorithmic trick, is going to be the solution to the problem of learning hierarchical procedures. We believe that this is a cognitive science problem that probably isn't amenable to a purely computer science oriented solution. In other words, we suspect that the correct way to break a Combo tree down into hierarchical components depends on context, algorithms are of course required, but they're algorithms for relating a CT to its context rather than pure CT-manipulation algorithms. Dimensional embedding is arguably a tool for capturing contextual relationships, but it's a very crude one.

Generally speaking, what we need to be learning are patterns of the form "A subtree meeting requirements X is often fit when linked to a subtree meeting requirements Y, when solving a problem of type Z". Here the context requirements Y will not pertain to absolute tree position but rather to abstract properties of a subtree.

The MetaNode approach as outlined above is a kind of halfway measure toward this goal, good because of its relative computational efficiency, but ultimately too limited in its power to deal with really hard hierarchical learning problems. The reason the MetaNode approach is crude is simply because it involves describing subtrees via points in an embedding space. We believe that the *correct* (but computationally expensive) approach is indeed to use MetaNodes - but with each MetaNode tagged, not with coordinates in an embedding space, but with a set of logical relationships describing the subtree that the MetaNode stands for. A candidate subtree's similarity to the MetaNode may then be determined by inference rather than by the simple computation of a distance between points in the embedding space. (And, note that we may have a hierarchy of MetaNodes, with small subtrees corresponding to MetaNodes, larger subtrees comprising networks of small subtrees also corresponding to MetaNodes, etc.)

The question then becomes which logical relationships one tries to look for, when characterizing a MetaNode. This may be partially domain-specific,

in the sense that different properties will be more interesting when studying motor-control procedures than when studying cognitive procedures.

To intuitively understand the nature of this idea, let's consider some abstract but commonsense examples. Firstly, suppose one is learning procedures for serving a ball in tennis. Suppose all the successful procedures work by first throwing the ball up really high, then doing other stuff. The internal details of the different procedures for throwing the ball up really high may be wildly different. What we need is to learn the pattern

```
Implication
    Inheritance X ``throwing the ball up really high''
    ``X then Y'' is fit
```

Here X and Y are MetaNodes. But the question is how do we learn to break trees down into MetaNodes according to the formula "tree='X then Y' where X inherits from 'throwing the ball up really high."'?

Similarly, suppose one is learning procedures to do first-order inference. What we need is to learn a pattern such as:

```
Implication
    AND
        F involves grabbing pairs from the AtomTable
        G involves applying an inference rule to those each pair
        H involves putting the results back in the AtomTable
    ``F ( G (H)))'' is fit
```

Here we need MetaNodes for F, G and H, but we need to characterize e.g. the MetaNode F by a relationship such as "involves grabbing pairs from the AtomTable."

Until we can characterize MetaNodes using abstract descriptors like this, one might argue we're just doing "statistical learning" rather than "general intelligence style" procedure learning. But to do this kind of abstraction intelligently seems to require some background knowledge about the domain.

In the "throwing the ball up really high" case the assignment of a descriptive relationship to a subtree involves looking, not at the internals of the subtree itself, but at the state of the world after the subtree has been executed.

In the "grabbing pairs from the AtomTable" case it's a bit simpler but still requires some kind of abstract model of what the subtree is doing, i.e. a model involving a logic expression such as "The output of F is a set S so that if P belongs to S then P is a set of two Atoms A1 and A2, and both A1 and A2 were produced via the getAtom operator."

How can this kind of abstraction be learned? It seems unlikely that abstractions like this will be found via evolutionary search over the space of all possible predicates describing program subtrees. Rather, they need to be found via probabilistic reasoning based on the terms combined in subtrees, put together with background knowledge about the domain in which the fitness function exists. In short, integrative cognition is required to

learn hierarchically structured programs in a truly effective way, because the appropriate hierarchical breakdowns are contextual in nature, and to search for appropriate hierarchical breakdowns without using inference to take context into account, involves intractably large search spaces. WIKISOURCE:FitnessEstimationViaIntegrativeIntelligence

33.6 Fitness Function Estimation via Integrative Intelligence

If instance generation is very cheap and fitness evaluation is very expensive (as is the case in many applications of evolutionary learning in CogPrime), one can accelerate evolutionary learning via a "fitness function estimation" approach. Given a fitness function embodied in a predicate P, the goal is to learn a predicate Q so that:

1. Q is much cheaper than P to evaluate, and
2. There is a high-strength relationship:

```
Similarity Q P
```

or else

```
ContextLink C (Similarity Q P)
```

where C is a relevant context.

Given such a predicate Q, one could proceed to optimize P by ignoring evolutionary learning altogether and just repeatedly following the algorithm:

- Randomly generate N candidate solutions.
- Evaluate each of the N candidate solutions according to Q.
- Take the k≪N solutions that satisfy Q best, and evaluate them according to P.

improved based on the new evaluations of P that are done. Of course, this would not be as good as incorporating fitness function estimation into an overall evolutionary learning framework.

Heavy utilization of fitness function estimation may be appropriate, for example, if the entities being evolved are schemata intended to control an agent's actions in a real or simulated environment. In this case the specification predicate P, in order to evaluate P(S), has to actually use the schema S to control the agent in the environment. So one may search for Q that do not involve any simulated environment, but are constrained to be relatively small predicates involving only cheap-to-evaluate terms (e.g. one may allow standard combinators, numbers, strings, ConceptNodes, and predicates built up recursively from these). Then Q will be an abstract predictor of concrete environment success.

We have left open the all-important question of how to find the 'specification approximating predicate' Q.

One approach is to use evolutionary learning. In this case, one has a population of predicates, which are candidates for Q. The fitness of each candidate Q is judged by how well it approximates P over the set of candidate solutions for P that have already been evaluated. If one uses evolutionary learning to evolve Q's, then one is learning a probabilistic model of the set of Q's, which tries to predict what sort of Q's will better solve the optimization problem of approximating P's behavior. Of course, using evolutionary learning for this purpose potentially initiates an infinite regress, but the regress can be stopped by, at some level, finding Q's using a non-evolutionary learning based technique such as genetic programming, or a simple evolutionary learning based technique like standard BOA programming.

Another approach to finding Q is to use inference based on background knowledge. Of course, this is complementary rather than contradictory to using evolutionary learning for finding Q. There may be information in the knowledge base that can be used to "analogize" regarding which Q's may match P. Indeed, this will generally be the case in the example given above, where P involves controlling actions in a simulated environment but Q does not.

An important point is that, if one uses a certain Q1 within fitness estimation, the evidence one gains by trying Q1 on numerous fitness cases may be utilized in future inferences regarding other Q2 that may serve the role of Q. So, once inference gets into the picture, the quality of fitness estimators may progressively improve via ongoing analogical inference based on the internal structures of the previously attempted fitness estimators.

Section X
Declarative Learning

Chapter 34
Probabilistic Logic Networks

Co-authored with Matthew Ikle'

34.1 Introduction

Now we turn to CogPrime 's methods for handling declarative knowledge – beginning with a series of chapters discussing the Probabilistic Logic Networks (PLN) [GMIH08] approach to uncertain logical reasoning, and then turning to chapters on pattern mining and concept creation. In this first of the chapters on PLN, we give a high-level overview, summarizing material given in the book *Probabilistic Logic Networks* [GMIH08] more compactly and in a somewhat differently-organized way. For a more thorough treatment of the concepts and motivations underlying PLN, the reader is encouraged to read [GMIH08].

PLN is a mathematical and software framework for uncertain inference, operative within the CogPrime software framework and intended to enable the combination of probabilistic truth values with general logical reasoning rules. Some of the key requirements underlying the development of PLN were the following:

- To enable uncertainty-savvy versions of all known varieties of logical reasoning, including for instance higher-order reasoning involving quantifiers, higher-order functions, and so forth
- To reduce to crisp "theorem prover" style behavior in the limiting case where uncertainty tends to zero
- To encompass inductive and abductive as well as deductive reasoning
- To agree with probability theory in those reasoning cases where probability theory, in its current state of development, provides solutions within reasonable calculational effort based on assumptions that are plausible in the context of real-world embodied software systems
- To gracefully incorporate heuristics not explicitly based on probability theory, in cases where probability theory, at its current state of development, does not provide adequate pragmatic solutions

- To provide "scalable" reasoning, in the sense of being able to carry out inferences involving billions of premises.
- To easily accept input from, and send input to, natural language processing software systems In practice, PLN consists of
- a set of inference rules (e.g. deduction, Bayes rule, variable unification, modus ponens, etc.), each of which takes one or more logical relationships or terms (represented as CogPrime Atoms) as inputs, and produces others as outputs
- specific mathematical formulas for calculating the probability value of the conclusion of an inference rule based on the probability values of the premises plus (in some cases) appropriate background assumptions.

PLN also involves a particular approach to estimating the confidence values with which these probability values are held (weight of evidence, or second-order uncertainty). Finally, the implementation of PLN in software requires important choices regarding the structural representation of inference rules, and also regarding "inference control" – the strategies required to decide what inferences to do in what order, in each particular practical situation. Currently PLN is being utilized to enable an animated agent to achieve goals via combining actions in a game world. For example, it can figure out that to obtain an object located on top of a wall, it may want to build stairs leading from the floor to the top of the wall. Earlier PLN applications have involved simpler animated agent control problems, and also other domains, such as reasoning based on information extracted from biomedical text using a language parser.

For all its sophistication, however, PLN falls prey to the same key weakness as other logical inference systems: combinatorial explosion. In trying to find a logical chain of reasoning leading to a desired conclusion, or to evaluate the consequences of a given set of premises, PLN may need to explore an unwieldy number of possible combinations of the Atoms in CogPrime 's memory. For PLN to be practical beyond relatively simple and constrained problems (and most definitely, for it to be useful for AGI at the human level or beyond), it must be coupled with a powerful method for "inference tree pruning" – for paring down the space of possible inferences that the PLN engine must evaluate as it goes about its business in pursuing a given goal in a certain context. Inference control will be addressed in Chapter 36.

34.2 First Order Probabilistic Logic Networks

We now review the essentials of PLN in a more formal way. PLN is divided into first-order and higher-order sub-theories (FOPLN and HOPLN). These terms are used in a nonstandard way drawn conceptually from NARS [Wan06]. We develop FOPLN first, and then derive HOPLN therefrom.

FOPLN is a term logic, involving terms and relationships (links) between terms. It is an uncertain logic, in the sense that both terms and relationships are associated with truth value objects, which may come in multiple varieties ranging from single numbers to complex structures like indefinite probabilities. Terms may be either elementary observations, or abstract tokens drawn from a token-set T.

34.2.1 Core FOPLN Relationships

"Core FOPLN" involves relationships drawn from the set: negation; Inheritance and probabilistic conjunction and disjunction; Member and fuzzy conjunction and disjunction. Elementary observations can have only Member links, while token terms can have any kinds of links. PLN makes clear distinctions, via link type semantics, between probabilistic relationships and fuzzy set relationships. Member semantics are usually fuzzy relationships (though they can also be crisp), whereas Inheritance relationships are probabilistic, and there are rules governing the interoperation of the two types.

Suppose a virtual agent makes an elementary VisualObservation o of a creature named Fluffy. The agent might classify o as belonging, with degree 0.9, to the fuzzy set of furry objects. The agent might also classify o as belonging with degree 0.8 to the fuzzy set of animals. The agent could then build the following links in its memory:

Member o furry $< 0.9 >$
Member o animals $< 0.8 >$

The agent may later wish to refine its knowledge, by combining these MemberLinks. Using the minimum fuzzy conjunction operator, the agent would conclude:

fuzzyAND $< 0.8 >$
  Member o furry
  Member o animals

meaning that the observation o is a visual observation of a fairly furry, animal object.

The semantics of (extensional) Inheritance are quite different from, though related to, those of the MemberLink. ExtensionalInheritance represents a purely conditional probabilistic subset relationship and is represented through the Subset relationship. If A is Fluffy and B is the set of cat, then the statement

Subset $< 0.9 >$
  A

B

means that

$$P(x \text{ is in the set } B | x \text{ is in the set } A) = 0.9.$$

34.2.2 PLN Truth Values

PLN is equipped with a variety of different types of truth-value types. In order of increasing information about the full probability distribution, they are:

- strength truth-values, which consist of single numbers; e.g., $< s >$ or $< .8 >$. Usually strength values denote probabilities but this is not always the case.
- SimpleTruthValues, consisting of pairs of numbers. These pairs come in two forms: $< s,\ w >$, where s is a strength and w is a "weight of evidence" and $< s,\ N >$, where N is a "count." "Weight of evidence" is a qualitative measure of belief, while "count" is a quantitative measure of accumulated evidence.
- IndefiniteTruthValues, which quantify truth-values in terms of an interval $[L, U]$, a credibility level b, and an integer k (called the lookahead). IndefiniteTruthValues quantify the idea that after k more observations there is a probability b that the conclusion of the inference will appear to lie in $[L, U]$.
- DistributionalTruthValues, which are discretized approximations to entire probability distributions.

34.2.3 Auxiliary FOPLN Relationships

Beyond the core FOPLN relationships, FOPLN involves additional relationship types of two varieties. There are simple ones like Similarity, defined by

$$Similarity\ A\ B$$

We say a relationship R is simple if the truth value of $R\ A\ B$ can be calculated solely in terms of the truth values of core FOPLN relationships between A and B. There are also complex "auxiliary" relationships like Intensional-Inheritance, which as discussed in depth in the Appendix G, measures the extensional inheritance between the set of properties or patterns associated with one term and the corresponding set associated with another.

Returning to our example, the agent may observe that two properties of cats are that they are furry and purr. Since the Fluffy is also a furry animal, the agent might then obtain, for example

IntensionalInheritance $< 0.5 >$
  Fluffy
  cat

meaning that the Fluffy shares about 50% of the properties of cat. Building upon this relationship even further, PLN also has a mixed intensional/extensional Inheritance relationship which is defined simply as the disjunction of the Subset and IntensionalInheritance relationships.

As this example illustrates, for a complex auxiliary relationship R, the truth value of $R\ A\ B$ is defined in terms of the truth values of a number of different FOPLN relationships among different terms (others than A and B), specified by a certain mathematical formula.

34.2.4 PLN Rules and Formulas

A distinction is made in PLN between rules and formulas. PLN logical inferences take the form of "syllogistic rules," which give patterns for combining statements with matching terms. Examples of PLN rules include, but are not limited to,

- deduction $((A \to B) \wedge (B \to C) \Rightarrow (A \to C))$,
- induction $((A \to B) \wedge (A \to C) \Rightarrow (B \to C))$,
- abduction $((A \to C) \wedge (B \to C) \Rightarrow (A \to C))$,
- revision, which merges two versions of the same logical relationship that have different truth values
- inversion $((A \to B) \Rightarrow (B \to A))$.

The basic schematic of the first four of these rules is shown in Figure 34.1. We can see that the first three rules represent the natural ways of doing inference on three interrelated terms. We can also see that induction and abduction can be obtained from the combination of deduction and inversion, a fact utilized in PLN's truth value formulas.

Related to each rule is a formula which calculates the truth value resulting from application of the rule. As an example, suppose s_A, s_B, s_C, s_{AB}, and s_{BC} represent the truth values for the terms A, B, C, as well the truth values of the relationships $A \to B$ and $B \to C$, respectively. Then, under suitable conditions imposed upon these input truth values, the formula for the deduction rule is given by:

$$s_{AC} = s_{AB}s_{BC} + \frac{(1 - s_{AB})\,(s_C - s_B s_{BC})}{1 - s_B},$$

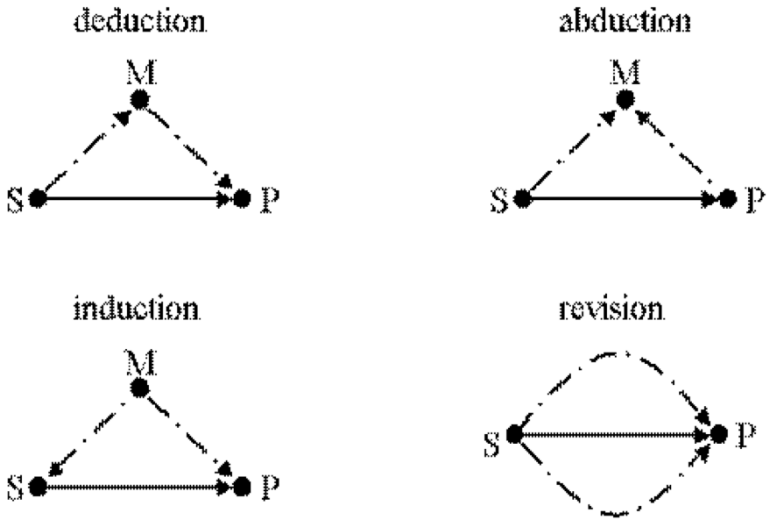


Fig. 34.1 The four most basic first-order PLN inference rules

where s_{AC} represents the truth value of the relationship $A \to C$. This formula is directly derived from probability theory given the assumption that $A \to B$ and $B \to C$ are independent.

For inferences involving solely fuzzy operators, the default version of PLN uses standard fuzzy logic with min/max truth value formulas (though alternatives may also be employed consistently with the overall PLN framework). Finally, the semantics of combining fuzzy and probabilistic operators is hinted at in [GMIH08] but addressed more rigorously in [GL10], which give a precise semantics for constructs of the form

$$\textit{Inheritance } A \ B$$

where A and B are characterized by relationships of the form *Member* C A, *Member* D B, etc. The various PLN rules have been gathered in appendix [TODO APPENDIX]

It is easy to see that, in the crisp case, where all MemberLinks and InheritanceLinks have strength 0 or 1, FOPLN reduces to standard propositional logic. Where inheritance is crisp but membership isn't, FOPLN reduces to higher-order fuzzy logic (including fuzzy statements about terms or fuzzy statements, etc.).

34.3 Higher-Order PLN

Higher-order PLN (HOPLN) is defined as the subset of PLN that applies to predicates (considered as functions mapping arguments into truth values). It includes mechanisms for dealing with variable-bearing expressions and higher-order functions.

A predicate, in PLN, is a special kind of term that embodies a function mapping terms or relationships into truth-values. HOPLN contains several relationships that act upon predicates including Evaluation, Implication, and several types of quantifiers. The relationships can involve constant terms, variables, or a mixture.

The Evaluation relationship, for example, evaluates a predicate on an input term. An agent can thus create a relationship of the form

Evaluation
    near
    (Bob's house, Fluffy)

or, as an example involving variables,

Evaluation
    near
    (X, Fluffy)

The Implication relationship is a particularly simple kind of HOPLN relationship in that it behaves very much like FOPLN relationships, via substitution of predicates in place of simple terms. Since our agent knows, for example,

Implication
    is_Fluffy
    AND is_furry purrs

and

Implication
    AND is_furry purrs
    is_cat

the agent could then use the deduction rule to conclude

Implication is_Fluffy is_cat

PLN supports a variety of quantifiers, including traditional crisp and fuzzy quantifiers, plus the AverageQuantifier defined so that the truth value of

$$AverageQuantifier\ X\ F(X)$$

is a weighted average of $F(X)$ over all relevant inputs X. AverageQuantifier is used implicitly in PLN to handle logical relationships between predicates, so that e.g. the conclusion of the above deduction is implicitly interpreted as

AverageQuantifier X
  Implication
    Evaluation is_Fluffy X
    Evaluation is_cat X

We can now connect PLN with the SRAM model (defined in Chapter 7 of Part 1).

Suppose for instance that the agent observes Fluffy from across the room, and that it has previously learned a Fetch procedure that tells it how to obtain an entity once it sees that entity. Then, if the agent has the goal of finding a cat, and it has concluded based on the above deduction that Fluffy is indeed a cat (since it is observed to be furry and purr), the cognitive schematic (knowledge of the form *Context* & *Procedure* $\rightarrow$ *Goal* as explained in Chapter 8 of Part 1) may suggest it to execute the Fetch procedure.

34.3.1 Reducing HOPLN to FOPLN

In [GMIH08] it is shown that in principle, over any finite observation set, HOPLN reduces to FOPLN. The key ideas of this reduction are the elimination of variables via use of higher-order functions, and the use of the set-theoretic definition of function embodied in the *SatisfyingSet* operator to map function-argument relationships into set-member relationships.

As an example, consider the Implication link. In HOPLN, where X is a variable

Implication
  R_1 A X
  R_2 B X

may be reduced to

Inheritance
  SatisfyingSet(R_1 A X)
  SatisfyingSet(R_2 B X)

where e.g. *SatisfyingSet*(R_1 A X) is the fuzzy set of all X satisfying the relationship $R_1(A, X)$.

Furthermore in Appendix G, we show how experience-based possible world semantics can be used to reduce PLN's existential and universal quantifiers

to standard higher order PLN relationships using AverageQuantifier relationships. This completes the reduction of HOPLN to FOPLN in the SRAM context.

One may then wonder why it makes sense to think about HOPLN at all. The answer is that it provides compact expression of a specific subset of FOPLN expressions, which is useful in cases where agents have limited memory and these particular expressions provide agents practical value (it biases the agent's reasoning ability to perform just as well as in first or higher orders).

34.4 Predictive Implication and Attraction

This section briefly reviews the notions of predictive implication and predictive attraction, which are critical to many aspects of CogPrime dynamics including goal-oriented behavior.

Define

```
Attraction A B <s>
```

as P(B|A) - P(B|¬A) = s, or in node and link terms

```
s = (Inheritance A B).s - (Inheritance ¬A B).s
```

For instance

```
(Attraction fat pig).s =
(Inheritance fat pig).s - (Inheritance ¬fat pig).s
```

Relatedly, in the temporal domain, we have the link type PredictiveImplication, where

```
PredictiveImplication A B <s>
```

roughly means that s is the probability that

```
Implication A B <s>
```

holds and also A occurs before B. More sophisticated versions of PredictiveImplication come along with more specific information regarding the time lag between A and B: for instance a time interval T in which the lag must lie, or a probability distribution governing the lag between the two events.

We may then introduce

```
PredictiveAttraction A B <s>
```

to mean

```
s = (PredictiveImplication A B).s - (PredictiveImplication ¬A B).s
```

For instance

```
(PredictiveAttraction kiss_Ben be_happy).s =
(PredictiveImplication kiss_Ben be_happy).s
- (PredictiveImplication ¬kiss_Ben be_happy).s
```

This is what really matters in terms of determining whether kissing Ben is worth doing in pursuit of the goal of being happy, not just how likely it is

to be happy if you kiss Ben, but how differentially likely it is to be happy if you kiss Ben.

Along with predictive implication and attraction, sequential logical operations are important, represented by operators such as SequentialAND, SimultaneousAND and SimultaneousOR. For instance:

```
PredictiveAttraction
    SequentialAND
        Teacher says 'fetch'
        I get the ball
        I bring the ball to the teacher
    I get a reward
```

combines SequentialAND and PredictiveAttraction. In this manner, an arbitrarily complex system of serial and parallel temporal events can be constructed.

34.5 Confidence Decay

PLN is all about uncertain truth values, yet there is an important kind of uncertainty it doesn't handle explicitly and completely in its standard truth value representations: the decay of information with time.

PLN does have an elegant mechanism for handling this: in the <s,d> formalism for truth values, strength s may remain untouched by time (except as new evidence specifically corrects it), but d may decay over time. So, our confidence in our old observations decreases with time. In the indefinite probability formalism, what this means is that old truth value intervals get wider, but retain the same mean as they had back in the good old days.

But the tricky question is: How fast does this decay happen?

This can be highly context-dependent.

For instance, 20 years ago we learned that the electric guitar is the most popular instrument in the world, and also that there are more bacteria than humans on Earth. The former fact is no longer true (keyboard synthesizers have outpaced electric guitars), but the latter is. And, if you'd asked us 20 years ago which fact would be more likely to become obsolete, we would have answered the former - because we knew particulars of technology would likely change far faster than basic facts of earthly ecology.

On a smaller scale, it seems that estimating confidence decay rates for different sorts of knowledge in different contexts is a tractable data mining problem, that can be solved via the system keeping a record of the observed truth values of a random sampling of Atoms as they change over time. (Operationally, this record may be maintained in parallel with the SystemActivityTable and other tables maintained for purposes of effort estimation, attention allocation and credit assignment.) If the truth values of a certain

sort of Atom in a certain context change a lot, then the confidence decay rate for Atoms of that sort should be increased.

This can be quantified nicely using the indefinite probabilities framework.

For instance, we can calculate, for a given sort of Atom in a given context, separate b-level credible intervals for the L and U components of the Atom's truth value at time t-r, centered about the corresponding values at time t. (This would be computed by averaging over all t values in the relevant past, where the relevant past is defined as some particular multiple of r; and over a number of Atoms of the same sort in the same context.)

Since historically-estimated credible-intervals won't be available for every exact value of r, interpolation will have to be used between the values calculated for specific values of r.

Also, while separate intervals for L and U would be kept for maximum accuracy, for reasons of pragmatic memory efficiency one might want to maintain only a single number x, considered as the radius of the confidence interval about both L and U. This could be obtained by averaging together the empirically obtained intervals for L and U.

Then, when updating an Atom's truth value based on a new observation, one performs a revision of the old TV with the new, but before doing so, one first **widens** the interval for the old one by the amounts indicated by the above-mentioned credible intervals.

For instance, if one gets a new observation about A with TV (L_{new}, U_{new}), and the prior TV of A, namely (L_{old}, U_{old}), is 2 weeks old, then one may calculate that L_{old} should really be considered as:

```
$(L_{old} - x, L_{old}+x)$
```

and U_old should really be considered as:

```
$(U_{old} - x, U_{old} + x)$
```

so that (L_new, U_new) should actually be revised with:

```
$(L_{old} - x, U_{old} + x)$
```

to get the total:

```
(L,U)
```

for the Atom after the new observation.

Note that we have referred fuzzily to "sort of Atom" rather than "type of Atom" in the above. This is because Atom type is not really the right level of specificity to be looking at. Rather - as in the guitar vs. bacteria example above - confidence decay rates may depend on semantic categories, not just syntactic (Atom type) categories. To give another example, confidence in the location of a person should decay more quickly than confidence in the location of a building. So ultimately confidence decay needs to be managed by a pool of learned predicates, which are applied periodically. These predicates are mainly to be learned by data mining, but inference may also play a role in some cases.

The ConfidenceDecay MindAgent must take care of applying the confidence-decaying predicates to the Atoms in the AtomTable, periodically.

The ConfidenceDecayUpdater MindAgent must take care of:

- forming new confidence-decaying predicates via data mining, and then revising them with the existing relevant confidence-decaying predicates.
- flagging confidence-decaying predicates which pertain to important Atoms but are unconfident, by giving them STICurrency, so as to make it likely that they will be visited by inference.

34.5.1 An Example

As an example of the above issues, consider that the confidence decay of:

```
Inh Ari male
```

should be low whereas that of:

```
Inh Ari tired
```

should be higher, because we know that for humans, being male tends to be a more permanent condition than being tired.

This suggests that concepts should have context-dependent decay rates, e.g. in the context of humans, the default decay rate of maleness is low whereas the default decay rate of tired-ness is high.

However, these defaults can be overridden. For instance, one can say "As he passed through his 80's, Grandpa just got tired, and eventually he died." This kind of tiredness, even in the context of humans, does not have a rapid decay rate. This example indicates why the confidence decay rate of a particular Atom needs to be able to override the default.

In terms of implementation, one mechanism to achieve the above example would be as follows. One could incorporate an *interval* confidence decay rate as an optional component of a truth value. As noted above one can keep two separate intervals for the L and U bounds; or to simplify things one can keep a single interval and apply it to both bounds separately.

Then, e.g., to define the decay rate for tiredness among humans, we could say:

```
ImplicationLink_HOJ
   InheritanceLink $X human
   InheritanceLink $X tired <confidenceDecay = [0,.1]>
```

or else (preferably):

```
ContextLink
  human
  InheritanceLink $X tired <confidenceDecay = [0,.1]>
```

Similarly, regarding maleness we could say:

```
ContextLink
  human
  Inh $X male <confidenceDecay = [0,.00001]>
```

Then one way to express the violation of the default in the case of grandpa's tiredness would be:

```
InheritanceLink grandpa tired <confidenceDecay = [0,.001]>
```

(Another way to handle the violation from default, of course, would be to create a separate Atom:

```
tired_from_old_age
```

and consider this as a separate sense of "tired" from the normal one, with its own confidence decay setting.)

In this example we see that, when a new Atom is created (e.g. *InheritanceLink Ari tired*), it needs to be assigned a confidence decay rate via inference based on relations such as the ones given above (this might be done e.g. by placing it on the queue for immediate attention by the ConfidenceDecayUpdater MindAgent). And periodically its confidence decay rate could be updated based on ongoing inferences (in case relevant abstract knowledge about confidence decay rates changes). Making this sort of inference reasonably efficient might require creating a special index containing abstract relationships that tell you something about confidence decay adjustment, such as the examples given above.

34.6 Why is PLN a Good Idea?

We have explored the intersection of the family of conceptual and formal structures that is PLN, with a specific formal model of intelligent agents (SRAM) and its extension using the cognitive schematic. The result is a simple and explicit formulation of PLN as a system by which an agent can manipulate tokens in its memory, thus represent observed and conjectured relationships (between its observations and between other relationships), in a way that assists it in choosing actions according to the cognitive schematic.

We have *not*, however, rigorously answered the question: What is the contribution of PLN to intelligence, within the formal agents framework introduced above? This is a quite subtle question, to which we can currently offer only an intuitive answer, not a rigorous one.

Firstly, there is the question of whether probability theory is really the best way to manage uncertainty, in a practical context. Theoretical results like those of Cox [**?**] and de Finetti [**?**] demonstrate that probability theory is the optimal way to handle uncertainty, if one makes certain reasonable assumptions. However, these reasonable assumptions don't actually apply to real-world intelligent systems, which must operate with relatively severe computational resource constraints. For example, one of Cox's axioms dictates

that a reasoning system must assign the same truth value to a statement, regardless of the route it uses to derive the statement. This is a nice idealization, but it can't be expected of any real-world, finite-resources reasoning system dealing with a complex environment. So an open question exists, as to whether probability theory is actually the best way for practical AGI systems to manage uncertainty. Most contemporary AI researchers assume the answer is yes, and probabilistic AI has achieved increasing popularity in recent years. However, there are also significant voices of dissent, such as Pei Wang [**?**] in the AGI community, and many within the fuzzy logic community.

PLN is not strictly probabilistic, in the sense that it combines formulas derived rigorously from probability theory with others that are frankly heuristic in nature. PLN was created in a spirit of open-mindedness regarding whether probability theory is actually the optimal approach to reasoning under uncertainty using limited resources, versus merely an approximation to the optimal approach in this case. Future versions of PLN might become either more or less strictly probabilistic, depending on theoretical and practical advances.

Next, aside from the question of the practical value of probability theory, there is the question of whether PLN in particular is a good approach to carrying out significant parts of what an AGI system needs to do, to achieve human-like goals in environments similar to everyday human environments.

Within a cognitive architecture where explicit utilization the cognitive schematic (Context & Procedure $\rightarrow$ Goal) is useful, clearly PLN is useful if it works reasonably well – so this question partially reduces to: what are the environments in which agents relying on the cognitive schematic are intelligent according to formal intelligent measures like those defined in Chapter 7 of Part 1. And then there is the possibility that some uncertain reasoning formalism besides PLN could be even more useful in the context of the cognitive schematic.

In particular, the question arises: What are the unique, peculiar aspects of PLN that makes it more useful in the context of the cognitive schematic, than some other, more straightforward approach to probabilistic inference? Actually there are multiple such aspects that we believe make it particularly useful. One is the indefinite probability approach to truth values, which we believe is more robust for AGI than known alternatives. Another is the clean reduction of higher order logic (as defined in PLN) to first-order logic (as defined in PLN), and the utilization of term logic instead of predicate logic wherever possible – these aspects make PLN inferences relatively simple in most cases where, according to human common sense, they should be simple.

A relatively subtle issue in this regard has to do with PLN intension. The cognitive schematic is formulated in terms of PredictiveExtensionalImplication (or any other equivalent way like PredictiveExtensionalAttraction), which means that intensional PLN links are not required for handling it. The hypothesis of the usefulness of intensional PLN links embodies a subtle assumption about the nature of the environments that intelligent agents are operating in. As discussed in [Goe06a] it requires an assumption related to

Peirce's philosophical axiom of the "tendency to take habits," which posits that in the real world, entities possessing some similar patterns have a probabilistically surprising tendency to have more similar patterns.

Reflecting on these various theoretical subtleties and uncertainties, one may get the feeling that the justification for applying PLN in practice is quite insecure! However, it must be noted that no other formalism in AI has significantly better foundation, at present. Every AI method involves certain heuristic assumptions, and the applicability of these assumptions in real life is nearly always a matter of informal judgment and copious debate. Even a very rigorous technique like a crisp logic formalism or support vector machines for classification, requires non-rigorous heuristic assumptions to be applied to the real world (how does sensation and actuation get translated into logic formulas, or SVM feature vectors)? It would be great if it were possible to use rigorous mathematical theory to derive an AGI design, but that's not the case right now, and the development of this sort of mathematical theory seems quite a long way off. So for now, we must proceed via a combination of mathematics, practice and intuition.

In terms of demonstrated practical utility, PLN has not yet confronted any really ambitious AGI-type problems, but it has shown itself capable of simple practical problem-solving in areas such as virtual agent control [**?**] and natural language based scientific reasoning [**?**]. The current PLN implementation within CogPrime can be used to learn to play fetch or tag, draw analogies based on observed objects, or figure out how to carry out tasks like finding a cat. We expect that further practical applications, as well as very ambitious AGI development, can be successfully undertaken with PLN without a theoretical understanding of exactly what are the properties of the environments and goals involved that allow PLN to be effective. However, we expect that a deeper theoretical understanding may enable various aspects of PLN to be adjusted in a more effective manner.

Chapter 35
Spatiotemporal Inference

35.1 Introduction

Most of the problems and situations humans confront every day involve space and time explicitly and centrally. Thus, any AGI system aspiring to humanlike general intelligence must have some reasonably efficient and general capability to solve spatiotemporal problems. Regarding how this capability might get into the system, there is a spectrum of possibilities, ranging from rigid hard-coding to tabula rasa experiential learning. Our bias in this regard is that it's probably sensible to somehow "wire into" CogPrime some knowledge regarding space and time – these being, after all, very basic categories for any embodied mind confronting the world.

It's arguable whether the explicit insertion of prior knowledge about space-time is *necessary* for achieving humanlike AGI using feasible resources. As an argument *against* the necessity of this sort of prior knowledge, Ben Kuipers and his colleagues [SMK12] have shown that an AI system can learn via experience that its perceptual stream comes from a world with three, rather than two or four dimensions. There is a long way from learning the number of dimensions in the world to learning the full scope of practical knowledge needed for effectively reasoning about the world – but it does seem plausible, from their work, that a broad variety of spatiotemporal knowledge could be inferred from raw experiential data. On the other hand, it also seems clear that the human brain does not do it this way, and that a rich fund of spatiotemporal knowledge is "hard-coded" into the brain by evolution – often in ways so low-level that we take them for granted, e.g. the way some motion detection neurons fire in the physical direction of motion, and the way somatosensory cortex presents a distorted map of the body's surface. On a psychological level, it is known that some fundamental intuition for space and time is hard-coded into the human infant's brain [Joh05]. So while we consider the learning of basic spatiotemporal knowledge from raw experience a worthy research direction, and fully compatible with the CogPrime vision;

yet for our main current research, we have chosen to hard-wire some basic spatiotemporal knowledge.

If one does wish to hard-wire some basic spatiotemporal knowledge into one's AI system, multiple alternate or complementary methodologies may be used to achieve this, including spatiotemporal logical inference, internal simulation, or techniques like recurrent neural nets whose dynamics defy simple analytic explanation. Though our focus in this chapter is on inference, we must emphasize that inference, even very broadly conceived, is not the only way for an intelligent agent to solve spatiotemporal problems occurring in its life. For instance, if the agent has a detailed map of its environment, it may be able to answer some spatiotemporal questions by directly retrieving information from the map. Or, logical inference may be substituted or augmented by (implicitly or explicitly) building a model that satisfies the initial knowledge – either abstractly or via incorporating "visualization" connected to sensory memory – and then interpret new knowledge over that model instead of inferring it. The latter is one way to interpret what DeSTIN and other CSDLNs do; indeed, DeSTIN's perceptual hierarchy is often referred to as a "state inference hierarchy." Any CSDLN contains biasing toward the commonsense structure of space and time, in its spatiotemporal hierarchical structure. It seems plausible that the human mind uses a combination of multiple methods for spatiotemporal understanding, just as we intend CogPrime to do.

In this chapter we focus on spatiotemporal logical inference, addressing the problem of creating a spatiotemporal logic adequate for use within an AGI system that confronts the same sort of real-world problems that humans typically do. The idea is not to fully specify the system's understanding of space and time in advance, but rather to provide some basic spatiotemporal logic rules, with parameters to be adjusted based on experience, and the opportunity for augmenting the logic over time with experientially-acquired rules. Most of the ideas in this chapter are reviewed in more detail, with more explanation, in the book *Real World Reasoning* [GCG+11]; this chapter represent a concise summary, compiled with the AGI context specifically in mind.

A great deal of excellent work has already been done in the areas of spatial, temporal and spatiotemporal reasoning; however, this work does not quite provide an adequate foundation for a logic-incorporating AGI system to do spatiotemporal reasoning, because it does not adequately incorporate uncertainty. Our focus here is to extend existing spatiotemporal calculi to appropriately encompass uncertainty, which we argue is sufficient to transform them into an AGI-ready spatiotemporal reasoning framework. We also find that a simple extension of the standard PLN uncertainty representations, inspired by $\mathcal{P}(\mathcal{Z})$-logic [Yan10], allows more elegant expression of probabilistic fuzzy predicates such as arise naturally in spatiotemporal logic.

In the final section of the chapter, we discuss the problem of **planning**, which has been considered extensively in the AI literature. We describe an

approach to planning that incorporates PLN inference using spatiotemporal logic, along with MOSES as a search method, and some record-keeping methods inspired by traditional AI planning algorithms.

35.2 Related Work on Spatio-temporal Calculi

We now review several calculi that have previously been introduced for representing and reasoning about space, time and space-time combined.

Spatial Calculi

Calculi dealing with space usually model three types of relationships between spatial regions: topological, directional and metric.

The most popular calculus dealing with topology is the Region Connection Calculus (RCC) [RCC93], relying on a base relationship `C` (for `Connected`) and building up other relationships from it, like `P` (for `PartOf`), or `O` (for `Overlap`). For instance $\texttt{P}(X, Y)$, meaning X is a part of Y, can be defined using `C` as follows

$$\texttt{P}(X, Y) \text{ iff } \forall Z \in \mathcal{U}, \texttt{C}(Z, X) \implies \texttt{C}(Z, Y) \tag{35.1}$$

Where $\mathcal{U}$ is the universe of regions. RCC-8 models eight base relationships, see Figure 35.1. And it is possible, using the notion of convexity, to model

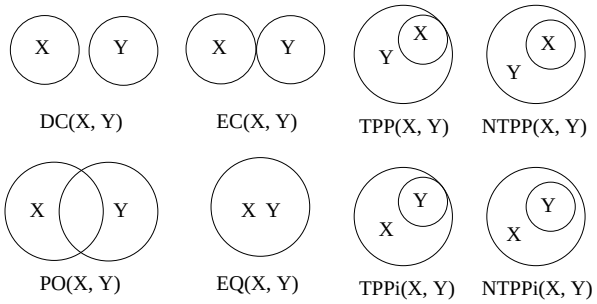


Fig. 35.1 The eight base relationships of RCC-8

more relationships such as inside, partially inside and outside, see Figure 35.2. For instance RCC-23 [Ben94] is an extension of RCC-8 using relationships based on the notion of convexity. The 9-intersection calculus [Win95, Kur09] is another calculus for reasoning on topological relationships, but handling relationships between heterogeneous objects, points, lines, surfaces.

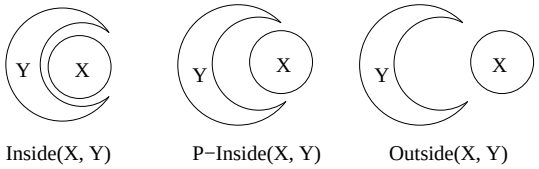


Fig. 35.2 Additional relationships using convexity

Regarding reasoning about direction, the Cardinal Direction Calculus [GE01, ZLLY08] considers directional relationships between regions, to express propositions such as "region A is to the north of region B".

And finally regarding metric reasoning, spatial reasoning involving qualitative distance (such as close, medium, far) and direction combined is considered in [CFH97].

Some work has also been done to extend and combine these various calculi, such as combining RCC-8 and the Cardinal Direction Calculus [LLR09], or using size [GR00] or shape [Coh95] information in RCC.

Temporal Calculi

The best known temporal calculus is Allen's Interval Algebra [All83], which considers 13 relationships over time intervals, such as `Before`, `During`, `Overlap`, `Meet`, etc. For instance one can express that digestion occurs after or right after eating by

$$\texttt{Before}(\mathit{Eat}, \mathit{Digest}) \vee \texttt{Meet}(\mathit{Eat}, \mathit{Digest})$$

equivalently denoted $\mathit{Eat}\{\texttt{Before}, \texttt{Meet}\}\mathit{Digest}$. There also exists a generalization of Allen's Interval Algebra that works on semi-intervals [FF92], that are intervals with possibly undefined start or end.

There are modal temporal logics such as LTL and CTL, mostly used to check temporal constraints on concurrent systems such as deadlock or fairness using Model Checking [Mai00].

Calculi with Space and Time Combined

There exist calculi combining space and time, first of all those obtained by "temporizing" spatial calculus, that is tagging spatial predicates with timestamps or time intervals. For instance STCC (for Spatio-temporal Constraint Calculus) [GN02] is basically RCC-8 combined with Allen's Algebra. With STCC one can express spatiotemporal propositions such as

$$\texttt{Meet}(\texttt{DC}(Finger, Key), \texttt{EC}(Finger, Key))$$

which means that the interval during which the finger is away from the key meets the interval during which the finger is against the key.

Another way to combine space and time is by modeling motion; e.g. the Qualitative Trajectory Calculus (QTC) [WKB05] can be used to express whether 2 objects are going forward/backward or left/right relative to each other.

Uncertainty in Spatio-temporal Calculi

In many situations it is worthwhile or even necessary to consider non-crisp extensions of these calculi. For example it is not obvious how one should consider in practice whether two regions are connected or disconnected. A desk against the wall would probably be considered connected to it even if there is a small gap between the wall and the desk. Or if A is not entirely part of B it may still be valuable to consider to which extent it is, rather than formally rejecting $\texttt{PartOf}(A, B)$. There are several ways to deal with such phenomena; one way is to consider probabilistic or fuzzy extensions of spatiotemporal calculi.

For instance in [SDCCK08b, SDCCK08a] the RCC relationship `C` (for `Connected`) is replaced by a fuzzy predicate representing closeness between regions and all other relationships based on it are extended accordingly. So e.g. `DC` (for `Disconnected`) is defined as follows

$$\texttt{DC}(X, Y) = 1 - \texttt{C}(X, Y) \tag{35.2}$$

`P` (for `PartOf`) is defined as

$$\texttt{P}(X, Y) = \inf_{Z \in \mathcal{U}} I(\texttt{C}(Z, X), \texttt{C}(Z, Y)) \tag{35.3}$$

where I is a fuzzy implication with some natural properties (usually $I(x_1, x_2) = \max(1 - x_1, x_2)$). Or, `EQ` (for `Equal`) is defined as

$$\texttt{EQ}(X, Y) = \min(\texttt{P}(X, Y), \texttt{P}(Y, X)) \tag{35.4}$$

and so on.

However the inference rules cannot determine the exact fuzzy values of the resulting relationships but only a lower bound, for instance

$$T(\texttt{P}(X, Y), \texttt{P}(Y, Z)) \leq \texttt{P}(X, Z) \tag{35.5}$$

where $T(x_1, x_2) = \max(0, x_1 + x_2 - 1)$. This is to be expected since in order to know the resulting fuzzy value one would need to know the exact spatial

configuration. For instance Figure 35.3 depicts 2 possible configurations that would result in 2 different values of $\mathtt{P}(X, Z)$.

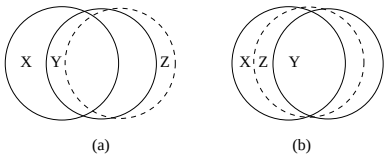


Fig. 35.3 Depending on where Z is, in dashline, $\mathtt{P}(X, Z)$ gets a different value.

One way to address this difficulty is to reason with interval-value fuzzy logic [DP09], with the downside of ending up with wide intervals. For example applying the same inference rule from Equation 35.5 in the case depicted in Figure 35.4 would result in the interval $[0, 1]$, corresponding to a state of total ignorance. This is the main reason why, as explained in the next section, we have decided to use distributional fuzzy values for our AGI-oriented spatiotemporal reasoning.

There also exist attempts to use probability with RCC. For instance in [Win00], RCC relationships are extracted from computer images and weighted based on their likelihood as estimated by a shape recognition algorithm. However, to the best of our knowledge, no one has used distributional fuzzy values [Yan10] in the context of spatiotemporal reasoning; and we believe this is important for the adaptation of spatiotemporal calculi to the AGI context.

35.3 Uncertainty with Distributional Fuzzy Values

$\mathcal{P}(\mathcal{Z})$ [Yan10] is an extension of fuzzy logic that considers distributions of fuzzy values rather than mere fuzzy values. That is, fuzzy connectors are extended to apply over probability density functions of fuzzy truth value. For instance the connector $\neg$ (often defined as $\neg x = 1 - x$) is extended such that the resulting distribution $\mu_\neg : [0, 1] \mapsto R^+$ is

$$\mu_\neg(x) = \mu(1 - x) \tag{35.6}$$

where μ is the probability density function of the unique argument. Similarly, one can define $\mu_\wedge : [0, 1] \mapsto R^+$ as the resulting density function of the connector $x_1 \wedge x_2 = \min(x_1, x_2)$ over the 2 arguments $\mu_1 : [0, 1] \mapsto R^+$ and $\mu_2 : [0, 1] \mapsto R^+$

$$\begin{aligned}\mu_{\wedge}(x) = \mu_1(x) \int_x^1 \mu_2(x_2)dx_2 \\ +\mu_2(x) \int_x^1 \mu_1(x_1)dx_1\end{aligned} \tag{35.7}$$

See [Yan10] for the justification of Equations 35.6 and 35.7.

Besides extending the traditional fuzzy operators, one can also define a wider class of connectors that can fully modulate the output distribution. Let $F : [0,1]^n \mapsto ([0,1] \mapsto \mathbb{R}^+)$ be a n-ary connector that takes n fuzzy values and returns a probability density function. In that case the probability density function resulting from the extension of F over distributional fuzzy values is:

$$\mu_F = \underbrace{\int_0^1 \dots \int_0^1}_{n} F(x_1, \dots, x_n)\mu_1(x_1) \dots \mu_n(x_n)dx_1 \dots dx_n \tag{35.8}$$

where μ_1, ..., μ_n are the n input arguments. That is, it is the average of all density functions output by F applied over all fuzzy input values. Let us call that type of connectors *fuzzy-probabilistic*.

In the following we give an example of such a fuzzy-probabilistic connector.

Example with `PartOf`

Let us consider the RCC relationship `PartOf` (`P` for short as defined in Equation 35.1). A typical inference rule in the crisp case would be:

$$\frac{\mathtt{P}(X,Y) \qquad \mathtt{P}(Y,Z)}{\mathtt{P}(X,Z)} \tag{35.9}$$

expressing the transitivity of `P`. But using a distribution of fuzzy values we would have the following rule

$$\frac{\mathtt{P}(X,Y)\ \langle\mu_1\rangle \qquad \mathtt{P}(Y,Z)\ \langle\mu_2\rangle}{\mathtt{P}(X,Z)\ \langle\mu_{POT}\rangle} \tag{35.10}$$

POT stands for PartOf Transitivity. The definition of μ_{POT} for that particular inference rule may depend on many assumptions like the shapes and sizes of regions X, Y and Z. In the following we will give an example of a definition of μ_{POT} with respect to some oversimplified assumptions chosen to keep the example short.

Let us define the fuzzy variant of $\mathtt{PartOf}(X,Y)$ as the proportion of X which is part of Y (as suggested in [Pal04]). Let us also assume that every region is a unitary circle. In this case, the required proportion depends solely

on the distance d_{XY} between the centers of X and Y, so we may define a function f that takes that distance and returns the according fuzzy value; that is, $f(d_{XY}) = \mathtt{P}(X, Y)$

$$f(d_{XY}) = \begin{cases} \dfrac{4\alpha - d_{XY}\sin(\alpha)}{2\pi} & \text{if } 0 \leq d_{XY} \leq 2 \\ 0 & \text{if } d_{XY} > 2 \end{cases} \tag{35.11}$$

where $\alpha = \cos^{-1}(d_{XY}/2)$.

For $0 \leq d_{XY} \leq 2$, $f(d_{XY})$ is monotone decreasing, so the inverse of $f(d_{XY})$, that takes a fuzzy value and returns a distance, is a function declared as $f^{-1}(x) : [0, 1] \mapsto [0, 2]$.

Let be $x_{XY} = \mathtt{P}(X, Y)$, $x_{YZ} = \mathtt{P}(Y, Z)$, $x = \mathtt{P}(X, Z)$, $d_{XY} = f^{-1}(x_{XY})$, $d_{YZ} = f^{-1}(x_{YZ})$, $l = |d_{XY} - d_{YZ}|$ and $u = d_{XY} + d_{YZ}$. For d_{XY} and d_{YZ} fixed, let $g : [0, \pi] \mapsto [l, u]$ be a function that takes as input the angle β of the two lines from the center of Y to X and Y to Z (as depicted in Figure 35.4) and returns the distance d_{XZ}. g is defined as follows

$$g(\beta) = \sqrt{(d_{XY} - d_{YZ}\sin(\beta))^2 + (d_{YZ}\cos(\beta))^2}$$

So $l \leq d_{XZ} \leq u$. It is easy to see that g is monotone increasing and surjective, therefore there exists a function inverse $g^{-1} : [l, u] \mapsto [0, \pi]$. Let $h = f \circ g$, so h

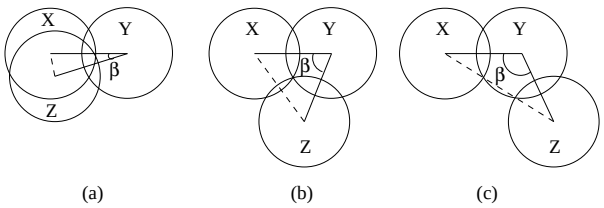


Fig. 35.4 d_{XZ}, in dashline, for 3 different angles

takes an angle as input and returns a fuzzy value, $h : [0, \pi] \mapsto [0, 1]$. Since f is monotone decreasing and g is monotone increasing, h is monotone decreasing. Note that the codomain of h is $[0, f^{-1}(l)]$ if $l < 2$ or $\{0\}$ otherwise. Assuming that $l < 2$, then the inverse of h is a function with the following signature $h^{-1} : [0, f^{-1}(l)] \mapsto [0, \pi]$. Using h^{-1} and assuming that the probability of picking $\beta \in [0, \pi]$ is uniform, we can define the binary connector POT. Let us define $\nu = POT(x_{XY}, x_{YZ})$, recalling that POT returns a density function and assuming $x < f^{-1}(l)$

$$\begin{aligned}\nu(x) &= 2 \lim_{\delta \to 0} \frac{\int_{h^{-1}(x+\delta)}^{h^{-1}(x)} \frac{1}{\pi} d\beta}{\delta} \\ &= \frac{2}{\pi} \lim_{\delta \to 0} \frac{h^{-1}(x) - h^{-1}(x+\delta)}{\delta} \\ &= -\frac{2h^{-1'}(x)}{\pi}\end{aligned} \tag{35.12}$$

where $h^{-1'}$ is the derivative of h^{-1}. If $x \geq f^{-1}(l)$ then $\nu(x) = 0$. For sake of simplicity the exact expressions of h^{-1} and $\nu(x)$ have been left out, and the case where one of the fuzzy arguments x_{XY}, x_{YZ} or both are null has not been considered but would be treated similarly assuming some probability distribution over the distances d_{XY} and d_{XZ}.

It is now possible to define μ_{POT} in rule 35.10 (following Equation 35.8)

$$\mu_{POT} = \int_0^1 \int_0^1 POT(x_1, x_2)\mu_1(x_1)\mu_2(x_2)dx_1dx_2 \tag{35.13}$$

Obviously, assuming that regions are unitary circles is crude; in practice, regions might be of very different shapes and sizes. In fact it might be so difficult to chose the right assumptions (and once chosen to define POT correctly), that in a complex practical context it may be best to start with overly simplistic assumptions and then learn POT based on the experience of the agent. So the agent would initially perform spatial reasoning not too accurately, but would improve over time by adjusting POT, as well as the other connectors corresponding to other inference rules.

It may also be useful to have more premises containing information about the sizes (e.g $\texttt{Big}(X)$) and shapes (e.g $\texttt{Long}(Y)$) of the regions, like

$$\frac{\texttt{B}(X)\ \langle\mu_1\rangle \quad \texttt{L}(Y)\ \langle\mu_2\rangle \quad \texttt{P}(X,Y)\ \langle\mu_3\rangle \quad \texttt{P}(Y,Z)\ \langle\mu_4\rangle}{\texttt{P}(X,Z)\ \langle\mu\rangle}$$

where `B` and `L` stand respectively for `Big` and `Long`.

Simplifying Numerical Calculation

Using probability density as described above is computationally expensive, and in many practical cases it's overkill. To decrease computational cost, several cruder approaches are possible, such as discretizing the probability density functions with a coarse resolution, or restricting attention to beta distributions and treating only their means and variances (as in [Yan10]).

The right way to simplify depends on the fuzzy-probabilistic connector involved and on how much inaccuracy can be tolerated in practice.

35.4 Spatio-temporal Inference in PLN

We have discussed the representation of spatiotemporal knowledge, including associated uncertainty. But ultimately what matters is what an intelligent agent can do with this knowledge. We now turn to uncertain reasoning based on uncertain spatiotemporal knowledge, using the integration of the above-discussed calculi into the Probabilistic Logic Networks reasoning system, an uncertain inference framework designed specifically for AGI and integrated into the OpenCog AGI framework.

We give here a few examples of spatiotemporal inference rules coded in PLN. Although the current implementation of PLN incorporates both fuzziness and probability it does not have a built-in truth value to represent distributional fuzzy values, or rather a distribution of distribution of fuzzy value, as this is how, in essence, confidence is represented in PLN. At that point, depending on design choice and experimentation, it is not clear whether we want to use the existing truth values and treat them as distributional truth values or implement a new type of truth value dedicated for that, so for our present theoretical purposes we will just call it *DF Truth Value*.

Due to the highly flexible HOJ formalism (Higher Order Judgment, explained in the PLN book in detail) we can express the inference rule for the relationship `PartOf` directly as Nodes and Links as follows

$$\begin{array}{l}
\texttt{ForAllLink \$X \$Y \$Z} \\
\quad \texttt{ImplicationLink_HOJ} \\
\quad\quad \texttt{ANDLink} \\
\quad\quad\quad \texttt{PartOf}(\$X, \$Y)\ \langle \texttt{tv}_1 \rangle \\
\quad\quad\quad \texttt{PartOf}(\$Y, \$Z)\ \langle \texttt{tv}_2 \rangle \\
\quad\quad \texttt{ANDLink} \\
\quad\quad\quad \texttt{tv}_3 = \mu_{POT}(\texttt{tv}_1, \texttt{tv}_2) \\
\quad\quad\quad \texttt{PartOf}(\$X, \$Z)\ \langle \texttt{tv}_3 \rangle
\end{array} \tag{35.14}$$

where μ_{POT} is defined in Equation 35.13 but extended over the domain of PLN DF Truth Value instead of $\mathcal{P}(\mathcal{Z})$ distributional fuzzy value. Note that $\texttt{PartOf}(\$X, \$Y)\ \langle \texttt{tv} \rangle$ is a shorthand for

$$\begin{array}{l}
\texttt{EvaluationLink}\ \langle \texttt{tv} \rangle \\
\quad \texttt{PartOf} \\
\quad \texttt{ListLink} \\
\quad\quad \$X \\
\quad\quad \$Y
\end{array} \tag{35.15}$$

and `ForAllLink $X $Y $Z` is a shorthand for

```
ForAllLink
    ListLink
        $X
        $Y
        $Z
```
(35.16)

Of course one advantage of expressive the inference rule directly in Nodes and Links rather than a built-in PLN inference rule is that we can use OpenCog itself to improve and refine it, or even create new spatiotemporal rules based on its experience. In the next 2 examples the fuzzy-probabilistic connectors are ignored, (so no DF Truth Value is indicated) but one could define them similarly to μ_{POT}.

First consider a temporal rule from Allen's Interval Algebra. For instance "if $\$I_1$ meets $\$I_2$ and $\$I_3$ is during $\$I_2$ then $\$I_3$ is after $\$I_1$" would be expressed as

```
ForAllLink $I1 $I2 $I3
    ImplicationLink
        ANDLink
            Meet($I1, $I2)
            During($I3, $I2)
        After($I3, $I1)
```
(35.17)

And a last example with a metric predicate could be "if $X is near $Y and $X is far from $Z then $Y is far from $Y"

```
ForAllLink $X $Y $Z
    ImplicationLink_HOJ
        ANDLink
            Near($X, $Y)
            Far($X, $Z)
        Far($Y, $Z)
```
(35.18)

That is only a small and partial illustrative example – for instance other rules may be used to specify that `Near` and `Far` and reflexive and symmetric.

35.5 Examples

The ideas presented here have extremely broad applicability; but for sake of concreteness, we now give a handful of examples illustrating applications to commonsense reasoning problems.

35.5.1 Spatiotemporal Rules

The rules provided here are reduced to the strict minimum needed for the examples:

1. At $T, if $X is inside $Y and $Y is inside $Z then $X is inside $Z
   ForAllLink $T $X $Y $Z
       ImplicationLink_HOJ
           ANDLink
               atTime($T, Inside($X, $Y))
               atTime($T, Inside($Y, $Z))
           atTime($T, Inside($X, $Z))
2. If a small object $X is over $Y and $Y is far from $Z then $X is far from $Z
   ForAllLink
       ImplicationLink_HOJ
           ANDLink
               Small($X)
               Over($X, $Y)
               Far($Y)
           Far($X)

That rule is expressed in a crisp way but again is to be understood in an uncertain way, although we haven't worked out the exact formulae.

35.5.2 The Laptop is Safe from the Rain

A laptop is over the desk in the hotel room, the desk is far from the window and we want assess to which extend the laptop is far from the window, therefore same from the rain.

Note that the truth values are ignored but each concept is to be understood as fuzzy, that is having a PLN Fuzzy Truth Value but the numerical calculation are left out.

We want to assess how much the Laptop is far from the window
Far(Window, Laptop)
Assuming the following

1. The laptop is small
   Small(Laptop)
2. The laptop is over the desk
   Over(Laptop, Desk)
3. The desk is far from the window
   Far(Desk, Window)

Now we can show an inference trail that lead to the conclusion, the numeric calculation are let for later.

1. using axioms 1, 2, 3 and PLN AND rule
   `ANDLink`
   `Small(Laptop)`
   `Over(Laptop, Desk)`
   `Far(Desk, Window)`
2. using spatiotemporal rule 2, instantiated with `$X` = `Laptop`, `$Y` = `Desk` and `$Z` = `Window`
   `ImplicationLink_HOJ`
   `ANDLink`
   `Small(Laptop)`
   `Over(Laptop, Desk)`
   `Far(Desk, Window)`
   `Far(Laptop, Window)`
3. using the result of previous step as premise with PLN implication rule
   `Far(Laptop, Window)`

35.5.3 Fetching the Toy Inside the Upper Cupboard

Suppose we know that there is a toy in an upper cupboard and near a bag, and want to assess to which extend climbing on the pillow is going to bring us near the toy.

Here are the following assumptions

1. The toy is near the bag and inside the cupboard. The pillow is near and below the cupboard
   `Near(toy, bag)` $\langle \mathtt{tv}_1 \rangle$
   `Inside(toy, cupboard)` $\langle \mathtt{tv}_2 \rangle$
   `Below(pillow, cupboard)` $\langle \mathtt{tv}_3 \rangle$
   `Near(pillow, cupboard)` $\langle \mathtt{tv}_4 \rangle$
2. The toy is near the bad inside the cupboard, how much the toy is near the edge of the cupboard?
   `ImplicationLink_HOJ`
   `ANDLink`
   `Near(toy, bag)` $\langle \mathtt{tv}_1 \rangle$
   `Inside(toy, cupboard)` $\langle \mathtt{tv}_2 \rangle$
   `ANDLink`
   $\mathtt{tv}_3 = \mu_{F_1}(\mathtt{tv}_1, \mathtt{tv}_2)$
   `Near(toy, cupboard_edge)` $\langle \mathtt{tv}_3 \rangle$
3. If I climb on the pillow, then shortly after I'll be on the pillow
   `PredictiveImplicationLink`
   `Climb_on(pillow)`
   `Over(self, pillow)`

4. If I am on the pillow near the edge of the cupboard how near am I from the toy?
   ImplicationLink_HOJ
       ANDLink
           Below(pillow, cupboard) $\langle tv_1 \rangle$
           Near(pillow, cupboard) $\langle tv_2 \rangle$
           Over(self, pillow) $\langle tv_3 \rangle$
           Near(toy, cupboard_edge) $\langle tv_4 \rangle$
       ANDLink
           $tv_5 = \mu_{F_2}(tv_1, tv_2, tv_3, tv_4)$
           Near(self, toy) $\langle tv_5 \rangle$

   The target theorem is "How near I am from the toy if I climb on the pillow."
   PredictiveImplicationLink
       Climb_on(pillow)
       Near(self, toy) $\langle ? \rangle$
   And the inference chain as follows

1. Axiom 2 with axiom 1
   Near(toy, cupboard_edge) $\langle tv_6 \rangle$
2. Step 1 with axiom 1 and 3
   PredictiveImplicationLink
       Climb_on(pillow)
       ANDLink
           Below(pillow, cupboard) $\langle tv_3 \rangle$
           Near(pillow, cupboard) $\langle tv_4 \rangle$
           Over(self, pillow) $\langle tv_7 \rangle$
           Near(toy, cupboard_edge) $\langle tv_6 \rangle$
3. Step 2 with axiom 4, target theorem: How near I am from the toy if I climb on the pillow
   PredictiveImplicationLink
       Climb_on(pillow)
       Near(self, toy) $\langle tv_9 \rangle$

35.6 An Integrative Approach to Planning

Planning is a major research area in the mainstream AI community, and planning algorithms have advanced dramatically in the last decade. However, the best of breed planning algorithms are still not able to deal with planning in complex environments in the face of a high level of uncertainty, which is the sort of situation routinely faced by humans in everyday life. Really powerful planning, we suggest, requires an approach different than any of the dedicated planning algorithms, involving spatiotemporal logic combined with a sophisticated search mechanism (such as MOSES).

It may be valuable (or even necessary) for an intelligent system involved in planning-intensive goals to maintain a specialized planning-focused data structure to guide general learning mechanisms toward more efficient learning in a planning context. But even if so, we believe planning must ultimately be done as a case of more general learning, rather than via a specialized algorithm.

The basic approach we suggest here is to

- use MOSES for the core plan learning algorithm. That is, MOSES would maintain a population of "candidate partial plans", and evolve this population in an effort to find effective complete plans.
- use PLN to help in the fitness evaluation of candidate partial plans. That is, PLN would be used to estimate the probability that a partial plan can be extended into a high-quality complete plan. This requires PLN to make heavy use of spatiotemporal logic, as described in the previous sections of this chapter.
- use a GraphPlan-style [BF97] planning graph, to record information about candidate plans, and to propagate information about mutual exclusion between actions. The planning graph maybe be used to help guide both MOSES and PLN.

In essence, the planning graph simply records different states of the world that may be achievable, with a high-strength PredictiveImplicationLink pointing between state X and Y if X can sensibly serve as a predecessor to X; and a low-strength (but potentially high-confidence) PredictiveImplicationLink between X and Y if the former excludes the latter. This may be a subgraph of the Atomspace or it may be separately cached; but in each case it must be frequently accessed via PLN in order for the latter to avoid making a massive number of unproductive inferences in the course of assisting with planning.

One can think of this as being a bit like PGraphPlan [BL99], except that

- MOSES is being used in place of forward or backward chaining search, enabling a more global search of the plan space (mixing forward and backward learning freely)
- PLN is being used to estimate the value of partial plans, replacing heuristic methods of value propagation

Regarding PLN, one possibility would be to (explicitly, or in effect) create a special API function looking something like

```
EstimateSuccessProbability(PartialPlan PP, Goal G)
```

(assuming the goal statement contains information about the time allotted to achieve the goal). The PartialPlan is simply a predicate composed of predicates linked together via temporal links such as PredictiveImplication and SimultaneousAND. Of course, such a function could be used within many non-MOSES approaches to planning also.

Put simply, the estimation of the success probability is "just" a matter of asking the PLN backward-chainer to figure out the truth value of a certain ImplicationLink, i.e.

```
PredictiveImplicationLink [time-lag T]
   EvaluationLink do PP
   G
```

But of course, this may be a very difficult inference without some special guidance to help the backward chainer. The GraphPlan-style planning graph could be used by PLN to guide it in doing the inference, via telling it what variables to look at, in doing its inferences. This sort of reasoning also requires PLN to have a fairly robust capability to reason about time intervals and events occurring therein (i.e., basic temporal inference).

Regarding MOSES, given a candidate plan, it could look into the planning graph to aid with program tree expansion. That is, given a population of partial plans, MOSES would progressively add new nodes to each plan, representing predecessors or successors to the actions already described in the plans. In choosing which nodes to add, it could be probabilistically biased toward adding nodes suggested by the planning graph.

So, overall what we have is an approach to doing planning via MOSES, with PLN for fitness estimation – but using a GraphPlan-style planning graph to guide MOSES's exploration of the neighborhood of partial plans, and to guide PLN's inferences regarding the success likelihood of partial plans.

Chapter 36
Adaptive, Integrative Inference Control

36.1 Introduction

The subtlest and most difficult aspect of logical inference is not the logical rule-set nor the management of uncertainty, but the *control* of inference: the choice of which inference steps to take, in what order, in which contexts. Without effective inference control methods, logical inference is an unscalable and infeasible approach to learning declarative knowledge. One of the key ideas underlying the CogPrime design is that inference control cannot effectively be handled by looking at logic alone. Instead, effective inference control must arise from the intersection between logical methods and other cognitive processes. In this chapter we describe some of the general principles used for inference control in the CogPrime design.

Logic itself is quite abstract and relatively (though not entirely) independent of the specific environment and goals with respect to which a system's intelligence is oriented. Inference control, however, is (among other things) a way of adapting a logic system to operate effectively with respect to a specific environment and goal-set. So, the reliance of CogPrime 's inference control methods on the integration between multiple cognitive processes, is a reflection of the foundation of CogPrime on the assumption (articulated in Chapter 9) that the relevant environment and goals embody interactions between world-structures and interaction-structures best addressed by these various processes.

36.2 High-Level Control Mechanisms

The PLN implementation in CogPrime is complex and lends itself to utilization via many different methods. However, a convenient way to think about it is in terms of three basic backward-focused query operations:

- **findtv**, which takes in an expression and tries to find its truth value.
- **findExamples**, which takes an expression containing variables and tries to find concrete terms to fill in for the variables.
- **createExamples**, which takes an expression containing variables and tries to create new Atoms to fill in for the variables, using *concept creation* heuristics as discussed in a later chapter, coupled with inference for evaluating the products of concept creation.

and one forward-chaining operation:

- **findConclusions**, which takes a set of Atoms and seeks to draw the most interesting possible set of conclusions via combining them with each other and with other knowledge in the AtomTable.

These inference operations may of course call themselves and each other recursively, thus creating lengthy chains of diverse inference.

Findtv is quite straightforward, at the high level of discussion adopted here. Various inference rules may match the Atom; in our current PLN implementation, loosely described below, these inference rules are executed by objects called Evaluators. In the course of executing findtv, a decision must be made regarding how much attention to allocate to each one of these Evaluator objects, and some choices must be made by the objects themselves - issues that involve processes beyond pure inference, and will be discussed later in this chapter. Depending on the inference rules chosen, findtv may lead to the construction of inferences involving variable expressions, which may then be evaluated via findExamples or createExamples queries.

The findExamples operation, on the other hand, sometimes reduces to a simple search through the AtomSpace. On the other hand, it can also be done in a subtler way. If the findExamples Evaluator wants to find examples of $X so that F($X), but can't find any, then its next step is to run another findExamples query, looking for $G so that

```
Implication $G F
```

and then running findExamples on G rather than F. But what if this findExamples query doesn't come up with anything? Then it needs to run a createExamples query on the same implication, trying to build a $G satisfying the implication.

Finally, forward-chaining inference (findConclusions) may be conceived of as a special heuristic for handling special kinds of findExample problems. Suppose we have K Atoms and want to find out what consequences logically ensue from these K Atoms, taken together. We can form the conjunction of the K Atoms (let's call it C), and then look for $D so that

```
Implication C $D
```

Conceptually, this can be approached via findExamples, which defaults to createExamples in cases where nothing is found. However, this sort of findExamples problem is special, involving appropriate heuristics for combining

the conjuncts contained in the expression C, which embody the basic logic of forward-chaining rather than backward-chaining inference.

36.2.1 The Need for Adaptive Inference Control

It is clear that in humans, inference control is all about context. We use different inference strategies in different contexts, and learning these strategies is most of what *learning to think* is all about. One might think to approach this aspect of cognition, in the CogPrime design, by introducing a variety of different inference control heuristics, each one giving a certain algorithm for choosing which inferences to carry out in which order in a certain context. (This is similar to what has been done within Cyc, for example `http:\cyc.com`.) However, in keeping with the *integrated intelligence* theme that pervades CogPrime , we have chosen an alternate strategy for PLN. We have one inference control scheme, which is quite simple, but which relies partially on structures coming from outside PLN proper. The requisite variety of inference control strategies is provided by variety in the non-PLN structures such as

- HebbianLinks existing in the AtomTable.
- Patterns recognized via pattern-mining in the corpus of prior inference trails

36.3 Inference Control in PLN

We will now describe the basic "inference control" loop of PLN in CogPrime . We will discuss it in the context of backward chaining; the case of forward chaining is very similar.

Given an expression to evaluate via inference (according to any one of the query operations mentioned above), there is a collection of Evaluator objects that matches the expression.

First, each Evaluator object makes a preliminary assessment regarding how likely it is to come up with decent results. This assessment is made based on some preliminary explorations of what Atoms it might use to draw its conclusions - and study of various links (including HebbianLinks) that exist relating to its actions on these Atoms (we will give some examples of this shortly), and information regarding what Evaluators have been useful in what prior inferences in related contexts (stored in a structure called the InferencePatternRepository, to be discussed below).

Then, the overall evaluation process looks at the assessments made by each Evaluator and decides how much computational resource to allocate to each Evaluator. This we may call the "Evaluator choice problem" of inference control.

Finally, each Evaluator chosen, then needs to make choices regarding which Atoms to use in its inference - and this choice must be made by use of existing links, and information in the InferencePatternRepository. This is the "Atom choice problem" of inference control.

As an example of the choices to be made by an individual Evaluator, consider that to evaluate (Inheritance A C) via a deduction-based Evaluator, some collection of intermediate nodes for the deduction must be chosen. In the case of higher-order deduction, each deduction may involve a number of complicated subsidiary steps, so perhaps only a single intermediate node will be chosen. This choice of intermediate nodes must also be made via context-dependent probabilities. In the case of other Evaluators besides deduction, other similar choices must be made.

So the basic inference algorithm we have discussed is basic backward-chaining inference, but aggressive and adaptive pruning using HebbianLinks and other existing knowledge is done at every stage.

36.3.1 The Evaluator Choice Problem as a Bandit Problem

The evaluator choice problem, as described above, is an example of a "multi-armed bandit problem" as commonly studied in statistics. The atom choice problem is also a bandit problem. Both of these problems can be approached via using standard "bandit problem" heuristics.

The paradigm bandit problem involves a slot machine ("multi-armed bandit") with N levers to pull, each of which may have a different odds of yielding a reward each time it is pulled. The player's goal is to maximize his earnings, but when he starts out playing with the bandit, he has no idea which levers lead to which levels of reward. So, at any given point, he must balance two factors:

- Exploitation: pulling the level that seems to give maximum reward, based on experience so far
- Exploration: trying out various levers to get a sample of their performance, so as to get more confident estimates of the reward level offered by each lever

Obviously, as more experience is gained, the bias should shift from exploration towards exploitation. There is a substantial body of mathematics regarding bandit problems, but most of the results prove inapplicable in real-world situations due to making inappropriate statistical assumptions. In practice, the two most common algorithms for solving bandit problems are:

- epsilon-greedy: spend $1 - epsilon$ of your time exploiting the best option found so far (with "best" defined in terms of expected reward), and epsilon of your time randomly exploring other options

- softmax: assign a probability to each option, using a heuristics formula based on thermodynamics that assigns higher probabilities to options that have proved more successful in the past, with an exponential scaling that favors successful options non-linearly over unsuccessful ones

The only modification we choose to make to these simple algorithms in the CogPrime context is to replace the probability with a product:

```
probability * weight_of_evidence
```

This evidence-weighted probability may be used within both epsilon-greedy and softmax.

Choosing an Evaluator for an inference step within PLN is a bandit problem, where prior experience is used to provide initial probabilities for the options (which are possible Evaluators rather than levers to pull), and then inferences done using each option are used to provide increasingly confident probabilities for each option. The default approach within CogPrime is softmax, though epsilon-greedy is also provided and may prove better in some contexts.

In Atom selection, the options (levers) are Atoms to use within an inference rule (an Evaluator).

It is important to note, however, that the statistical nature of the Atom-choice bandit problem is different from the statistics of the Evaluator-choice bandit problem, because there are not that many Evaluators, but there are a lot of Atoms. It would be possible to merge the two into a single bandit problem, whose options were (Evaluator, Atom-set) tuples, but this seems an inferior heuristic approach. The Evaluator bandit problem involves a relatively small number of options, which makes it more tractable. So we have chosen to break down the inference control process into two levels of bandit problems: Evaluator choice and Atom choice.

The complexity of the inference control problem can be well-understood by observing that each individual step poses two difficult statistical inference problems, one nested within the other! What allows pragmatic inferences to be achievable at all is, clearly, prior knowledge, which allows most of the bandit problems occurring in a complex inference to be "pre-solved" via assumption of prior probabilities. Normally, only a handful of difficult steps in an inference need to actually be studied statistically via numerous iterations of the epsilon-greedy or softmax algorithms. On the other hand, the first few inferences in an unfamiliar domain may not connect with the system's knowledge base of prior probabilities, and may thus need to be done in a much slower, more statistically thorough way.

36.3.2 Chains of Thought

An alternate solution for inference control might be to try to segregate rules into pipeline stages. Thus, one might have two sets of rules: A and B, and we know (either a-priori, or from experience) that no rule in set B will produce results until some rule in set A has already produced results. By structuring the rule sets into such a pipeline, one has an alternate, and very attractive solution to the inference control problem.

In fact, one would want to push this to a hierarchical extreme: so, if rule A5 from set A triggered, we might know that rules B3, B8 and B9 from set B were almost sure to follow, and that rules B1, B2, B4, etc almost never fired. And so on for set C. These kinds of correlations can be discovered via entropy/mutual-information techniques.

This sort of chain-of-thought processing would be most useful for processing data from input sources (*i.e.* from the environment, from reading text, from chat, from Virtual Reality interactions) where forward-chaining is the appropriate mechanism to transform data. Most processing for most situations would follow well-established, previously-learned chains of thought. Provided that chains of thought are not heavily branched (*i.e.* for any given rule in set A, only a small number of rules in set B follow, *etc.*), then the 'logical deduction' or data processing of input can be performed quite rapidly.

At the same time, one would need to have a learning mechanism running in the background, exploring other possible "chains of thought" to see if they might produce useful results.

Finally it is interesting to note that the format of these chains of though may entirely be coded as Nodes and Link in the AtomSpace, thus enabling OpenCog to reason about it own reasoning process.

WIKISOURCE:InferencePatternMining

36.4 Inference Pattern Mining

Among the data used to guide the solution of the Evaluator choice problem, the most important component is explicit information regarding which Evaluators have been useful in which contexts during past inferences.

This information is stored in CogPrime in a data repository called the InferencePatternRepository - which is, quite simply, a special "data table" containing inference trees and patterns recognized therein. An "inference tree" refers to a tree whose nodes, called InferenceTreeNodes, are Atoms (or generally Atom-versions, Atoms with truth value relative to a certain context), and whose links are inference steps (so each link is labeled with a certain Evaluator).

Note that, in many cases, PLN creates a variety of exploratory inference trees internally, in the context of doing a single inference. Most of these

inference trees will never be stored in the AtomTable, because they are unconfident and may not have produced extremely useful results. However, they should still be stored in the InferencePatternRepository. Ideally one would store all inference trees there. In a large CogPrime system this may not be feasible, but then a wide variety of trees should still be retained, including mainly successful ones but also a sampling of unsuccessful ones for purpose of comparison.

The InferencePatternRepository may then be used in two ways:

- An inference tree being actively expanded (i.e. utilized within the PLN inference system) may be compared to inference trees in the repository, in real time, for guidance. That is, if a node N in an inference tree is being expanded, then the repository can be searched for nodes similar to N, whose contexts (within their inference trees) are similar to the context of N within its inference tree. A study can then be made regarding which Evaluators and Atoms were most useful in these prior, similar inferences, and the results of this can be used to guide ongoing inference.
- Patterns can be extracted from the store of inference trees in the InferencePatternRepository, and stored separately from the actual inference trees (in essence, these patterns are inference subtrees with variables in place of some of their concrete nodes or links). An inference tree being expanded can then be compared to these patterns instead of, or in addition to, the actual trees in the repository. This provides greater efficiency in the case of common patterns among inference trees.

A reasonable approach may be to first check for inference patterns and see if there are any close matches; and if there are not, to then search for individual inference trees that are close matches.

Mining patterns from the repository of inference trees is a potentially highly computationally expensive operation, but this doesn't particularly matter since it can be run periodically in the background while inference proceeds at its own pace in the foreground, using the mined patterns. Algorithmically, it may be done either by exhaustive frequent-itemset-mining (as in the Apriori or Relim algorithms), or by stochastic greedy mining. These operations should be carried out by an InferencePatternMiner.

36.5 Hebbian Inference Control

One aspect of inference control is Evaluator choice, which is based on mining contextually relevant information from the InferencePatternRepository (see InferencePatternMining). But, what about the Atom choice aspect? This can in some cases be handled via the InferencePatternRepository as well, but it is less likely to serve the purpose than in the case of Evaluator choice. Evaluator choice is about finding structurally similar inferences in roughly

similar contexts, and using them as guidance. But Atom choice has a different aspect: it is also about what Atoms have tended to be related to the other Atoms involved in an inference, generically, not just in the context of prior inferences, but in the context of prior perceptions, cognition and actions in general.

Concretely, this means that Atom choice must make heavy use of HebbianLinks (see Chapter 23). The formation of HebbianLinks will be discussed in the following chapter, on attention allocation. Here it will suffice to get across the basic idea. The discussion of HebbianLinks here will hopefully serve to help you understand the motivation for the HebbianLink formation algorithm to be discussed later. Of course, inference is not the only user of HebbianLinks in the CogPrime system, but its use of HebbianLinks is somewhat representative. Figure 36.1 gives a simple illustrative example of the use of attention allocation, via HebbianLink, for PLN backward chaining.

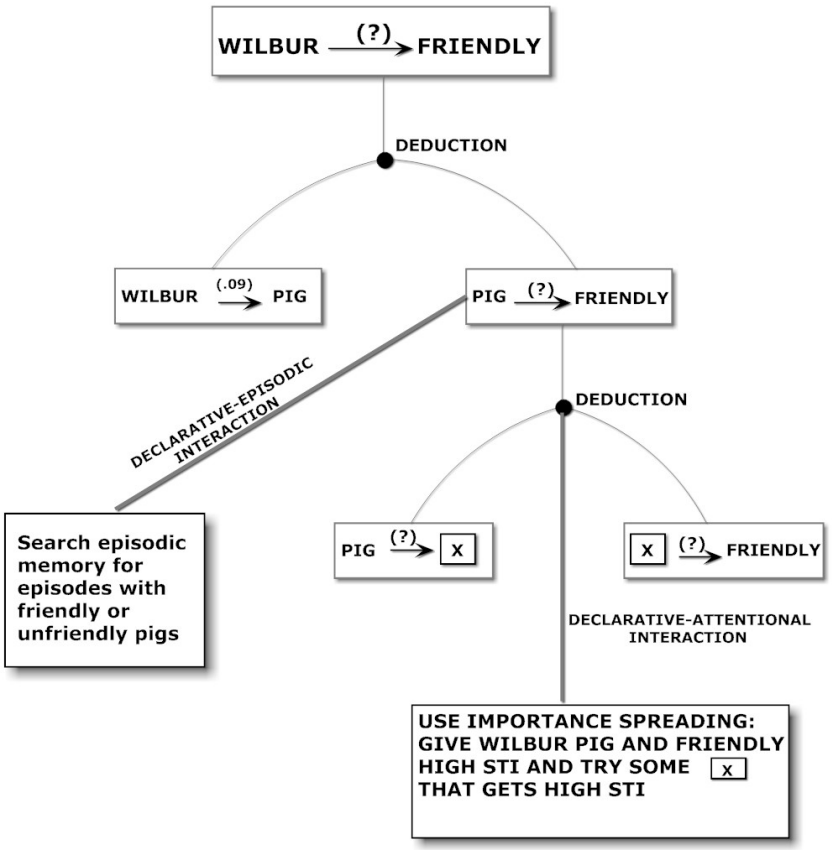


Fig. 36.1 The Use of Attention Allocation for Guiding Backward Chaining Inference.

The semantics of a HebbianLink between A and B is, intuitively: In the past, when A was important, B was also important. HebbianLinks are created via two basic mechanisms: pattern-mining of associations between importances in the system's history, and PLN inference based on HebbianLinks created via pattern mining (and inference). Thus, saying that PLN inference control relies largely on HebbianLinks is in part saying that PLN inference control relies on PLN. There is a bit of a recursion here, but it's not a bottomless recursion because it bottoms out with HebbianLinks learned via pattern mining.

As an example of the Atom-choices to be made by an individual Evaluator in the course of doing inference, consider that to evaluate (Inheritance A C) via a deduction-based Evaluator, some collection of intermediate nodes for the deduction must be chosen. In the case of higher-order deduction, each deduction may involve a number of complicated subsidiary steps, so perhaps only a single intermediate node will be chosen. This choice of intermediate nodes must be made via context-dependent prior probabilities. In the case of other Evaluators besides deduction, other similar choices must be made.

The basic means of using HebbianLinks in inferential Atom-choice is simple: If there are Atoms linked via HebbianLinks with the other Atoms in the inference tree, then these Atoms should be given preference in the Evaluator's (bandit problem based) selection process.

Along the same lines but more subtly, another valuable heuristic for guiding inference control is "on-the-fly associatedness assessment." If there is a chance to apply the chosen Evaluator via working with Atoms that are:

- strongly associated with the Atoms in the Atom being evaluated (via HebbianLinks)
- strongly associated with each other via HebbianLinks (hence forming a *cohesive set*)

then this should be ranked as a good thing.

For instance, it may be the case that, when doing deduction regarding relationships between humans, using relationships involving other humans as intermediate nodes in the deduction is often useful. Formally this means that, when doing inference of the form:

```
AND
  Inheritance A human
  Inheritance A B
  Inheritance C human
  Inheritance C B
|-
Inheritance A C
```

then it is often valuable to choose B so that:

```
HebbianLink B human
```

has high strength. This would follow from the above-mentioned heuristic.

Next, suppose one has noticed a more particular heuristic - that in trying to reason about humans, it is particularly useful to think about their wants. This suggests that in abductions of the above form it is often useful to choose B of the form:

```
B = SatisfyingSet [ wants(human, $X) ]
```

This is too fine-grained of a cognitive-control intuition to come from simple association-following. Instead, it requires fairly specific data-mining of the system's inference history. It requires the recognition of "Hebbian predicates" of the form:

```
HebbianImplication
  AND
    Inheritance $A human
    Inheritance $C human
    Similarity
      $B
      SatisfyingSet
         Evaluation wants (human, $X)
  AND
    Inheritance $A $B
    Inheritance $C $B
```

The semantics of:

```
HebbianImplication X Y
```

is that *when X is being thought about, it is often valuable to think about Y shortly thereafter.*

So what is required to do inference control according to heuristics like *think about humans according to their wants* is a kind of backward-chaining inference that combines Hebbian implications with PLN inference rules. PLN inference says that to assess the relationship between two people, one approach is abduction. But Hebbian learning says that when setting up an abduction between two people, one useful precondition is if the intermediate term in the abduction regards wants. Then a check can be made whether there are any relevant intermediate terms regarding wants in the system's memory.

What we see here is that the overall inference control strategy can be quite simple. For each Evaluator that can be applied, a check can be made for whether there is any relevant Hebbian knowledge regarding the general constructs involved in the Atoms this Evaluator would be manipulating. If so, then the prior probability of this Evaluator is increased, for the purposes of the Evaluator-choice bandit problem. Then, if the Evaluator is chosen, the specific Atoms this Evaluator would involve in the inference can be summoned up, and the relevant Hebbian knowledge regarding these Atoms can be utilized.

To take another similar example, suppose we want to evaluate:

```
Inheritance pig dog
```

via the deduction Evaluator (which also carries out induction and abduction). There are a lot of possible intermediate terms, but a reasonable heuristic is to ask a few basic questions about them: How do they move around? What do they eat? How do they reproduce? How intelligent are they? Some of these standard questions correspond to particular intermediate terms, e.g. the intelligence question partly boils down to computing:

```
Inheritance pig intelligent
```

and:

```
Inheritance dog intelligent
```

So a link:

```
HebbianImplication animal intelligent
```

may be all that's needed to guide inference to asking this question. This HebbianLink says that when thinking about animals, it's often interesting to think about intelligence. This should bias the deduction Evaluator to choose intelligent as an intermediate node for inference.

On the other hand, the *what do they eat* question is subtler and boils down to asking; Find $X so that when:

```
R($X) = SatisfyingSet[$Y] eats ($Y,$X)
```

holds (R($X) is a concept representing what eat $X), then we have:

```
Inheritance pig R($X)
```

and:

```
Inheritance dog R($X)
```

In this case, a HebbianLink from animal to eat would not really be fine-grained enough. Instead we want a link of the form:

```
HebbianImplication
    Inheritance $X animal
    SatisfyingSet[$Y] eats ($X, $Y)
```

This says that when thinking about an animal, it's interesting to think about what that animal eats.

The deduction Evaluator, when choosing which intermediate nodes to use, needs to look at the scope of available HebbianLinks and HebbianPredicates and use them to guide its choice. And if there are no good intermediate nodes available, it may report that it doesn't have enough experience to assess with any confidence whether it can come up with a good conclusion. As a consequence of the bandit-problem dynamics, it may be allocated reduced resources, or another Evaluator is chosen altogether.

36.6 Evolution As an Inference Control Scheme

It is possible to use PEPL (Probabilistic Evolutionary Program Learning) as, in essence, an InferenceControl scheme. Suppose we are using an evolutionary learning mechanism such as MOSES or PLEASURE [Goe08a] to evolve populations of predicates or schemata. Recall that there are two ways to evaluate procedures in CogPrime : by inference or by direct evaluation. Consider the case where inference is needed in order to provide high-confidence estimates of the evaluation or execution relationships involved. Then, there is the question of how much effort to spend on inference, for each procedure being evaluated as part of the fitness evaluation process. Spending a small amount of effort on inference means that one doesn't discover much beyond what's immediately apparent in the AtomSpace. Spending a large amount of effort on inference means that one is trying very hard to use indirect evidence to support conjectures regarding the evaluation or execution Links involved.

When one is evolving a large population of procedures, one can't afford to do too much inference on each candidate procedure being evaluated. Yet, of course, doing more inference may yield more accurate fitness evaluations, hence decreasing the number of fitness evaluations required.

Often, a good heuristic is to gradually increase the amount of inference effort spent on procedure evaluation, during the course of evolution. Specifically, one may make the amount of inference effort roughly proportional to the overall population fitness. This way, initially, evolution is doing a cursory search, not thinking too much about each possibility. But once it has some fairly decent guesses in its population, then it starts thinking hard, applying more inference to each conjecture.

Since the procedures in the population are likely to be interrelated to each other, inferences done on one procedure are likely to produce intermediate knowledge that's useful for doing inference on other procedures. Therefore, what one has in this scheme is evolution as a control mechanism for higher-order inference.

Combined with the use of evolutionary learning to achieve memory across optimization runs, this is a very subtle approach to inference control, quite different from anything in the domain of logic-based AI. Rather than guiding individual inference steps on a detailed basis, this type of control mechanism uses evolutionary logic to guide the general direction of inference, pushing the vast mass of exploratory inferences in the direction of solving the problem at hand, based on a flexible usage of prior knowledge.

36.7 Incorporating Other Cognitive Processes Into Inference

Hebbian inference control is a valuable and powerful process, but it is not always going to be enough. The solution of some problems that CogPrime chooses to address via inference will ultimately require the use of other methods, too. In these cases, one workaround is for inference to call on other cognitive processes to help it out.

This is done via the forward or backward chainer identifying specific Atoms deserving of attention by other cognitive processes, and then spawning Tasks executing these other cognitive processes on the appropriate Atoms.

Firstly, which Atoms should be selected for this kind of attention? What we want are InferenceTreeNodes that:

- have high STI.
- have the impact to significantly change the overall truth value of the inference tree they are embedded in (something that can be calculated by hypothetically varying the truth value of the InferenceTreeNode and seeing how the truth value of the overall conclusion is affected).
- have truth values that are known with low confidence.

Truth values meeting these criteria should be taken as strong candidates for attention by other cognitive processes.

The next question is which other cognitive processes do we apply in which cases?

MOSES in supervised categorization mode can be applied to a candidate InferenceTreeNode representing a CogPrime Node if it has a sufficient number of members (Atoms linked to it by MemberLinks); and, a sufficient number of new members have been added to it (or have had their membership degree significantly changed) since MOSES in supervised categorization mode was used on it last.

Next, pattern mining can be applied to look for connectivity patterns elsewhere in the AtomTable, similar to the connectivity patterns of the candidate Atom, if the candidate Atom has changed significantly since pattern mining last visited it.

More subtly, what if, we try to find whether "cross breed" implies "Ugliness", and we know that "bad genes" implies Ugliness, but can't find a way, by backward chaining, to prove that "cross breed" implies "bad genes". Then we could launch a non-backward-chaining algorithm to measure the overlap of SatisfyingSet(cross breed) and SatisfyingSet(bad genes). Specifically, we could use MOSES in supervised categorization mode to find relationships characterizing "cross breed" and other relationships characterizing "bad genes", and then do some forward chaining inference on these relationships. This would be a general heuristic for what to do when there's a link with low confidence but high potential importance to the inference tree.

SpeculativeConceptFormation (see Chapter 38) may also be used to create new concepts and attempt to link them to the Atoms involved in an inference (via subsidiary inference processes, or HebbianLink formation based on usage in learned procedures, etc.), so that they may be used in inference.

36.8 PLN and Bayes Nets

Finally, we give some comments on the relationship between PLN and Bayes Nets [PJ88a]. We have not yet implemented such an approach, but it may well be that Bayes Nets methods can serve as a useful augmentation to PLN for certain sorts of inference (specifically, for inference on networks of knowledge that are relatively static in nature).

We can't use standard Bayes Nets as the primary way of structuring reasoning in CogPrime because CogPrime 's knowledge network is loopy. The peculiarities that allow standard Bayes net belief propagation to work in standard loopy Bayes nets, don't hold up in CogPrime , because of the way you have to update probabilities when you're managing a very large network in interaction with a changing world, so that different parts of which get different amounts of focus. So in PLN we use different mechanisms (the "inference trail" mechanism) to avoid "repeated evidence counting" whereas in loopy Bayes nets they rely on the fact that in the standard loopy Bayes net configuration, extra evidence counting occurs in a fairly constant way across the network.

However, when you have within the AtomTable a set of interrelated knowledge items that you know are going to be static for a while, and you want to be able to query them probabilistically, then building a Bayes Net of some sort (i.e. "freezing" part of CogPrime 's knowledge network and mapping it into a Bayes Net) may be useful. I.e., one way to accelerate some PLN inference would be:

1. Freeze a subnetwork of the AtomTable which is expected not to change a lot in the near future
2. Interpret this subnetwork as a loopy Bayes net, and use standard Bayesian belief propagation to calculate probabilities based on it

This would be a highly efficient form of "background inference" in certain contexts. (Note that this requires an "indefinite Bayes net" implementation that propagates indefinite probabilities through the standard Bayes-net local belief propagation algorithms, but this is not problematic.)

Chapter 37
Pattern Mining

Co-authored with Jade O'Neill

37.1 Introduction

Having discussed inference in depth we now turn to other, simpler but equally important approaches to creating declarative knowledge. This chapters deals with pattern mining – the creation of declarative knowledge representing patterns among other knowledge (which may be declarative, sensory, episodic, procedural, etc.) – and the following chapter deals with speculative concept creation.

Within the scope of pattern mining, we will discuss two basic approaches:

- supervised learning: given a predicate, finding a pattern among the entities that satisfy that predicate.
- unsupervised learning: undirected search for "interesting patterns".

The supervised learning case is easier and we have done a number of experiments using MOSES for supervised pattern mining, on biological (microarray gene expression and SNP) and textual data. In the CogPrime case, the "positive examples" are the elements of the SatisfyingSet of the predicate P, and the "negative examples" are everything else. This can be a relatively straightforward problem if there are enough positive examples and they actually share common aspects ... but some trickiness emerges, of course, when the common aspects are, in each example, complexly intertwined with other aspects.

The unsupervised learning case is considerably trickier. The main problem issue here regards the definition of an appropriate fitness function. We are searching for "interesting patterns." So the question is, what constitutes an interesting pattern?

We will also discuss two basic algorithmic approaches:

- program learning, via MOSES or hillclimbing
- frequent subgraph mining, using greedy algorithms

The value of these various approaches is contingent on the environment and goal set being such that algorithms of this nature can actually recognize relevant patterns in the world and mind. Fortunately, the everyday human world does appear to have the property of possessing multiple relevant patterns that are recognizable using varying levels of sophistication and effort. It has patterns that can be recognized via simple frequent pattern mining, and other patterns that are too subtle for this, and are better addressed by a search-based approach. In order for an environment and goal set to be appropriate for the learning and teaching of a human-level AI, it should have the same property of possessing multiple relevant patterns recognizable using varying levels of subtlety.

37.2 Finding Interesting Patterns via Program Learning

As one important case of pattern mining, we now discuss the use of program learning to find "interesting" patterns in sets of Atoms.

Clearly, "interestingness" is a multidimensional concept. One approach to defining it is empirical, based on observation of which predicates have and have not proved interesting to the system in the past (based on their long-term importance values, i.e. LTI).

In this approach, one has a supervised categorization problem: learn a rule predicting whether a predicate will fall into the *interesting* category or the *uninteresting* category. Once one has learned this rule, which has expressed this rule as a predicate itself, one can then use this rule as the fitness function for evolutionary learning evolution.

There is also a simpler approach, which defines an *objective* notion of interestingness. This objective notion is a weighted sum of two factors:

- Compactness.
- Surprisingness of truth value.

Compactness is easy to understand: all else equal, a predicate embodied in a small Combo tree is better than a predicate embodied in a big one. There is some work hidden here in Combo tree reduction; ideally, one would like to find the smallest representation of a given Combo tree, but this is a computationally formidable problem, so one necessarily approaches it via heuristic algorithms.

Surprisingness of truth value is a slightly subtler concept. Given a Boolean predicate, one can envision two extreme ways of evaluating its truth value (represented by two different types of ProcedureEvaluator). One can use an IndependenceAssumingProcedureEvaluator, which deals with all AND and OR operators by assuming probabilistic independence. Or, one can use an ordinary EffortBasedProcedureEvaluator, which uses dependency information wherever feasible to evaluate the truth values of AND and OR opera-

tors. These two approaches will normally give different truth values but, how different? The more different, the more *surprising* is the truth value of the predicate, and the more *interesting* may the predicate be.

In order to explore the power of this kind of approach in a simple context, we have tested pattern mining using MOSES on Boolean predicates as a data mining algorithm on a number of different datasets, including some interesting and successful work in the analysis of gene expression data, and some more experimental work analyzing sociological data from the National Longitudinal Survey of Youth (NLSY) (http://stats.bls.gov/nls/).

A very simple illustrative result from the analysis of the NLSY data is the pattern:

```
OR
    (NOT(MothersAge(X)) AND NOT(FirstSexAge(X)))
    (Wealth(X) AND PIAT(X))
```

where the domain of X are individuals, meaning that:

- being the child of a young mother correlates with having sex at a younger age;
- being in a wealthier family correlates with better Math (PIAT) scores;
- the two sets previously described tend to be disjoint.

Of course, many data patterns are several times more complex than the simple illustrative pattern shown above. However, one of the strengths of the evolutionary learning approach to pattern mining is its ability to find simple patterns when they do exist, yet without (like some other mining methods) imposing any specific restrictions on the pattern format.

37.3 Pattern Mining via Frequent/Surprising Subgraph Mining

Probabilistic evolutionary learning is an extremely powerful approach to pattern mining, but, may not always be realistic due to its high computational cost. A cheaper, though also weaker, alternative, is to use frequent subgraph mining algorithms such as [HWP03, KK01], which may straightforwardly be adapted to hypergraphs such at the Atomspace.

Frequent subgraph mining is a port to the graph domain of the older, simpler idea of *frequent itemset mining*, which we now briefly review. There are a number of algorithms in the latter category, the classic is Apriori [AS94], and an alternative is relim [Bor05] which is conceptually similar but seems to give better performance.

The basic goal of frequent itemset mining is to discover frequent subsets ("itemsets") in a group of sets, whose members are all drawn from some base

set of items. One knows that for a set of N items, there are $2^N - 1$ possible subgroups. The algorithm operates in several rounds. Round i heuristically computes frequent i-itemsets (i.e. frequent sets containing i items). A round has two steps: candidate generation and candidate counting. In the candidate generation step, the algorithm generates a set of candidate i-itemsets whose support – the percentage of events in which the item must appear – has not been yet been computed. In the candidate-counting step, the algorithm scans its memory database, counting the support of the candidate itemsets. After the scan, the algorithm discards candidates with support lower than the specified minimum (an algorithm parameter) and retains only the sufficiently frequent i-itemsets. The algorithm reduces the number of tested subsets by pruning apriori those candidate itemsets that cannot be frequent, based on the knowledge about infrequent itemsets obtained from previous rounds. So for instance if $\{A, B\}$ is a frequent 2-itemset then $\{A, B, C\}$ will be considered as a potential 3-itemset, on the contrary if $\{A, B\}$ is not a frequent itemset then $\{A, B, C\}$, as well as any superset of $\{A, B\}$, will be discarded. Although the worst case of this sort of algorithm is exponential, practical executions are generally fast, depending essentially on the support limit.

Frequent subgraph mining follows the same pattern, but instead of a set of items it deals with a group of graphs. There are many frequent subgraph mining algorithms in the literature, but the basic concept underlying nearly all of them is the same: first find small frequent subgraphs. Then seek to find slightly larger frequent patterns encompassing these small ones. Then seek to find slightly larger frequent patterns encompassing *these*, etc. This approach is much faster than something like MOSES, although management of the large number of subgraphs to be searched through can require subtle design and implementation of data structures.

If, instead of an ensemble of small graphs, one has a single large graph like the AtomSpace, one can follow the same approach, via randomly subsampling from the large graph to find the graphs forming the ensemble to be mined from; see [ZH10] for a detailed treatment of this sort of approach. The fact that the AtomSpace is a hypergraph rather than a graph doesn't fundamentally affect the matter since a hypergraph may always be considered a graph via introduction of an additional node for each hyperedge (at the cost of a potentially great multiplication of the number of links).

Frequent subgraph mining algorithms appropriately deployed can find subgraphs which occur repeatedly in the Atomspace, including subgraphs containing Atom-valued variables . Each such subgraph may be represented as a PredicateNode, and frequent subgraph mining will find such PredicateNodes that have surprisingly high truth values when evaluated across the Atomspace. But unlike MOSES when applied as described above, such an algorithm will generally find such predicates only in a "greedy" way.

For instance, a greedy subgraph mining algorithm would be unlikely to find

```
OR
```

```
(NOT(MothersAge(X)) AND NOT(FirstSexAge(X)))
(Wealth(X) AND PIAT(X))
```

as a surprising pattern in an AtomSpace, unless at least one (and preferably both) of

```
Wealth(X) AND PIAT(X)
```

and

```
NOT(MothersAge(X)) AND NOT(FirstSexAge(X))
```

were surprising patterns in that Atomspace on their own.

37.4 Fishgram

Fishgram is an efficient algorithm for finding patterns in OpenCog knowledge, instantiating the general concepts presented in the previous section. It represents patterns as conjunctions (AndLink) of Links, which usually contain variables. It does a greedy search, so it can quickly find many patterns. In contrast, algorithms like MOSES are designed to find a small number of the best patterns. Fishgram works by finding a set of objects that have links in common, so it will be most effective if the AtomSpace has a lot of raw data, with simple patterns. For example, it can be used on the perceptions from the virtual world. There are predicates for basic perceptions (e.g. what kind of object something is, objects being near each other, types of blocks, and actions being performed by the user or the AI).

37.4.1 Example Patterns

Here is some example output from Fishgram, when run on the virtual agent's memories.

```
(AndLink
(EvaluationLink is_edible:PredicateNode (ListLink $1000041))
(InheritanceLink $1000041 Battery:ConceptNode)
)
```

This means a battery which can be "eaten" by the virtual robot. The variable $1000041 refers to the object (battery).

Fishgram can also find patterns containing a sequence of events. In this case, there is a list of EvaluationLinks or InheritanceLinks which describe the objects involved, followed by the sequence of events.

```
(AndLink
```

```
(InheritanceLink $1007703 Battery:ConceptNode)
(SequentialAndLink
(EvaluationLink isHolding:PredicateNode (ListLink $1008725 $1007703)))
)
)
```

This means the agent was holding a battery. $1007703 is the battery, and there is also a variables for the agent itself. Many interesting patterns involve more than one object. This pattern would also include the user (or another AI) holding a battery, because the pattern does not refer to the AI character specifically.

It can find patterns where it performs an action and achieves a goal. There is code to create implications based on these conjunctions. After finding many conjunctions, it can produce ImplicationLinks based on some of them. Here is an example where the AI-controlled virtual robot discovers how to get energy.

```
(ImplicationLink
(AndLink
(EvaluationLink is_edible:PredicateNode (ListLink $1011619))
(InheritanceLink $1011619 Battery:ConceptNode)
)
(PredictiveImplicationLink
(EvaluationLink actionDone:PredicateNode (ListLink (ExecutionLink eat:Grounded
(EvaluationLink increased:PredicateNode (ListLink (EvaluationLink
  EnergyDemandGoal:PredicateNode)))
)
)
```

37.4.2 The Fishgram Algorithm

The core Fishgram algorithm, in pseudocode, is as follows:

```
initial layer = every pair (relation, binding)

while previous layer is not empty:
foreach (conjunction, binding) in previous layer:
let incoming = all (relation, binding) pairs containing an object in the conju
let possible_next_events = all (event, binding) pairs where the event happens
foreach (relation, relation_binding) in incoming and possible_next_events:
new_relation = a copy of relation, where every variable that refer
if new_relation is already in conjunction, skip it
new_conjunction = conjunction + new_relation
if new_conjunction has been found already, skip it
otherwise, add new_conjunction to the current layer
```

```
map_to_existing_variables(conjunction, conjunction_binding, relation, relation
r', s' = a copy of the relation and binding using new variables
foreach variable v, object o in relation_binding:
foreach variable v2, object o2 in conjunction_binding:
if o == o2:
change r' and s' to use v2 instead of v
```

37.4.3 Preprocessing

There are several preprocessing steps to make it easier for the main Fishgram search to find patterns. There is a list of things that have to be variables. For example, any predicate that refers to object (including agents) will be given a variable so it can refer to any object. Other predicates or InheritanceLinks can be added to a pattern, to restrict it to specific kinds of objects, as shown above. So there is a step which goes through all of the links in the AtomSpace, and records a list of predicates with variables. Such as "X is red" or "X eats Y". This makes the search part simpler, because it never has to decide whether something should be a variable or a specific object.

There is also a filter system, so that things which seem irrelevant can be excluded from the search. There is a combinatorial explosion as patterns become larger. Some predicates may be redundant with each other, or known not to be very useful. It can also try to find only patterns in the AI's "attentional focus", which is much smaller than the whole AtomSpace.

The Fishgram algorithm cannot currently handle patterns involving numbers, although it could be extended to do so. The two options would be to either have a separate discretization step, creating predicates for different ranges of a value. Or alternatively, have predicates for mathematical operators. It would be possible to search for a "splitpoint" like in decision trees. So a number would be chosen, and only things above that value (or only things below that value) would count for a pattern. It would also be possible to have multiple numbers in a pattern, and compare them in various ways. It is uncertain how practical this would be in Fishgram. MOSES is good for finding numeric patterns, so it may be better to simply use those patterns inside Fishgram.

The "increased" predicate is added by a preprocessing step. The goals have a fuzzy TruthValue representing how well the goal is achieved at any point in time, so e.g. the EnergyDemandGoal represents how much energy the virtual robot has at some point in time. The predicate records times that a goal's TruthValue increased. This only happens immediately after doing something to increase it, which helps avoid finding spurious patterns.

37.4.4 Search Process

Fishgram search is breadth-first. It starts with all predicates (or InheritanceLinks) found by the preprocessing step. Then it finds pairs of predicates involving the same variable. Then they are extended to conjunctions of three predicates, and so on. Many relations apply at a specific time, for example the agent being near an object, or an action being performed. These are included in a sequence, and are added in the order they occurred.

Fishgram remembers the examples for each pattern. If there is only one variable in the pattern, an example is a single object; otherwise each example is a vector of objects for each variable in the pattern. Each time a relation is added to a pattern, if it has no new variables, some of the examples may be removed, because they don't satisfy the new predicate. It needs to have at least one variable in common with the previous relations. Otherwise the patterns would combine many unrelated things.

In frequent itemset mining (for example the APRIORI algorithm), there is effectively one variable, and adding a new predicate will often decrease the number of items that match. It can never increase it. The number of possible conjunctions increases with the length, up to some point, after which it decreases. But when mining for patterns with multiple objects there is a much larger combinatorial explosion of patterns. Various criteria can be used to prune the search.

The most basic criterion is the frequency. Only patterns with at least N examples will be included, where N is an arbitrary constant. You can also set a maximum number of patterns allowed for each length (number of relations), and only include the best ones. The next level of the breadth-first search will only search for extensions of those patterns.

One can also use a measure of statistical interestingness, to make sure the relations in a pattern are correlated with each other. There are many spurious frequent patterns, because anything which is frequent will occur together with other things, whether they are relevant or not. For example "breathing while typing" is a frequent pattern, because people breathe at all times. But "moving your hands while typing" is a much more interesting pattern. As people only move their hands some of the time, a measure of correlation would prefer the second pattern. The best measure may be interaction information, which is a generalisation of mutual information that applies to patterns with more than two predicates. An early-stage AI would not have much knowledge of cause and effect, so it would rely on statistical measures to find useful patterns.

37.4.5 Comparison to other algorithms

Fishgram is more suitable for OpenCogPrimes purposes than existing graph mining algorithms, most of which were designed with molecular datasets in

mind. The OpenCog AtomSpace is a different graph in various ways. For one, there are many possible relations between nodes (much like in a semantic network). Many relations involve more than two objects, and there are also properties Ð predicates about a single object. So the relations are effectively directed links of varying arity. It also has events, and many states can change over time (e.g. an egg changes state while it's cooking). Fishgram is designed for general knowledge in an embodied agent.

There are other major differences. Fishgram uses a breadth-first search, rather than depth-first search like most graph mining algorithms. And it does an "embedding-based" search, searching for patterns that can be embedded multiple times in a large graph. Molecular datasets have many separate graphs for separate molecules, but the embodied perceptions are closer to a single, fairly well-connected graph. Depth-first search would be very slow on such a graph, as there are many very long paths through the graph, and the search would mostly find those. Whereas the useful patterns tend to be compact and repeated many times.

Lastly the design of Fishgram makes it easy to experiment with multiple different scoring functions, from simple ones like frequency to much more sophisticated functions such as interaction information.

Chapter 38
Speculative Concept Formation

38.1 Introduction

One of the hallmarks of general intelligence is its capability to deal with novelty in its environment and/or goal-set. And dealing with novelty intrinsically requires creating novelty. It's impossible to efficiently handle new situations without creating new ideas appropriately. Thus, in any environment complex and dynamic enough to support human-like general intelligence (or any other kind of highly powerful general intelligence), the creation of novel ideas will be paramount. New idea creation takes place in OpenCog via a variety of methods – e.g. inside MOSES which creates new program trees, PLN which creates new logical relationships, ECAN which creates new associative relationships, etc. But there is also a role for explicit, purposeful creation of new Atoms representing new concepts, outside the scope of these other learning mechanisms.

The human brain gets by, in adulthood, without creating *that many* new neurons – although neurogenesis does occur on an ongoing basis. But this is achieved only via great redundancy, because for the brain it's cheaper to maintain a large number of neurons in memory at the same time, than to create and delete neurons. Things are different in a digital computer: memory is more expensive but creation and deletion of object is cheaper. Thus in CogPrime , forgetting and creation of Atoms is a regularly occurring phenomenon. In this chapter we discuss a key class of mechanisms for Atom creation, "speculative concept formation." Further methods will be discussed in following chapters.

The philosophy underlying CogPrime 's speculative concept formation is that new things should be created from pieces of good old things (a form of "evolution", broadly construed), and that probabilistic extrapolation from experience should be used to guide the creation of new things (inference). It's clear that these principles are necessary for the creation of new mental forms but it's not obvious that they're sufficient: this is a nontrivial hypothesis,

which may also be considered a family of hypotheses since there are many different ways to do extrapolation and intercombination. In the context of mind-world correspondence, the implicit assumption underlying this sort of mechanism is that the relevant patterns in the world can often be combined to form other relevant patterns. The everyday human world does quite markedly display this kind of combinatory structure, and such a property seems basic enough that it's appropriate for use as an assumption underlying the design of cognitive mechanisms.

In CogPrime we have introduced a variety of heuristics for creating new Atoms - especially ConceptNodes - which may then be reasoned on and subjected to implicit (via attention allocation) and explicit (via the application of evolutionary learning to predicates obtained from concepts via "concept predicatization") evolution. Among these are the node logical operators described in the PLN book, which allow the creation of new concepts via AND, OR, XOR and so forth. However, logical heuristics alone are not sufficient. In this chapter we will review some of the nonlogical heuristics that are used for speculative concept formation. These operations play an important role in creativity - to use cognitive-psychology language, they are one of the ways that CogPrime implements the process of blending, which Falconnier and Turner (2003) have argued is key to human creativity on many different levels. Each of these operations may be considered as implicitly associated with an hypothesis that, in fact, the everyday human world tends to assign utility to patterns that are combinations of other patterns produced via said operation.

An evolutionary perspective may also be useful here, on a technical level as well as philosophically. As noted in *The Hidden Pattern* and hinted Chapter in 3 of Part 1, one way to think about an AGI system like CogPrime is as a huge evolving ecology. The AtomSpace is a biosphere of sorts, and the mapping from Atom types into species has some validity to it (though not complete accuracy: Atom types do not compete with each other; but they do reproduce with each other, and according to most of the reproduction methods in use, Atoms of differing type cannot cross-reproduce). Fitness is defined by importance. Reproduction is defined by various operators that produce new Atoms from old, including the ones discussed in this chapter, as well as other operators such as inference and explicit evolutionary operators.

New ConceptNode creation may be triggered by a variety of circumstances. If two ConceptNodes are created for different purposes, but later the system finds that most of their meanings overlap, then it may be more efficient to merge the two into one. On the other hand, a node may become overloaded with different usages, and it is more useful to split it into multiple nodes, each with a more consistent content. Finally, there may be patterns across large numbers of nodes that merit encapsulation in individual nodes. For instance, if there are 1000 fairly similar ConceptNodes, it may be better not to merge them all together, but rather to create a single node to which they all link, reifying the category that they collectively embody.

In the following sections, we will begin by describing operations that create new ConceptNodes from existing ones on a local basis: by mutating individual ConceptNodes or combining pairs of ConceptNodes. Some of these operations are inspired by evolutionary operators used in the GA, others are based on the cognitive psychology concept of "blending." Then we will turn to the use of clustering and formal concept analysis algorithms inside CogPrime to refine the system's knowledge about existing concepts, and create new concepts.

38.2 Evolutionary Concept Formation

A simple and useful way to combine ConceptNodes is to use GA-inspired evolutionary operators: crossover and mutation. In mutation, one replaces some number of a Node's links with other links in the system. In crossover, one takes two nodes and creates a new node containing some links from one and some links from another.

More concretely, to cross over two ConceptNodes X and Y, one may proceed as follows (in short clustering the union of X and Y):

- Create a series of empty nodes $Z_1, Z_2, \ldots, Z_k$
- Form a "link pool" consisting of all X's links and all Y's links, and then divide this pool into clusters (clustering algorithms will be described below).
- For each cluster with significant cohesion, allocate the links in that cluster to one of the new nodes Z_i

On the other hand, to mutate a ConceptNode, a number of different mutation processes are reasonable. For instance, one can

- Cluster the links of a Node, and remove one or more of the clusters, creating a node with less links
- Cluster the links, remove one or more clusters, and then add new links that are similar to the links in the remaining clusters

The EvolutionaryConceptFormation MindAgent selects pairs of nodes from the system, where the probability of selecting a pair is determined by

- the average importance of the pair
- the degree of similarity of the pair
- the degree of association of the pair

(Of course, other heuristics are possible too). It then crosses over the pair, and mutates the result.

Note that, unlike in some GA implementations, the parent node(s) are retained within the system; they are not replaced by the children. Regardless of how many offspring they generate by what methods, and regardless of their

age, all Nodes compete and cooperate freely forever according to the fitness criterion defined by the importance updating function. The entire AtomSpace may be interpreted as a large evolutionary, ecological system, and the action of CogPrime dynamics, as a whole, is to create fit nodes.

A more advanced variant of the EvolutionaryConceptFormation MindAgent would adapt its mutation rate in a context-dependent way. But our intuition is that it is best to leave this kind of refinement for learned cognitive schemata, rather than to hard-wire it into a MindAgent. To encourage the formation of such schemata, one may introduce elementary schema functions that embody the basic node-level evolutionary operators:

```
ConceptNode ConceptCrossover(ConceptNode A, ConceptNode B)

ConceptNode mutate(ConceptNode A, mutationAmount m)
```

There will also be a role for more abstract schemata that utilize these. An example cognitive schema of this sort would be one that said: "When all my schema in a certain context seem unable to achieve their goals, then maybe I need new concepts in this context, so I should increase the rate of concept mutation and crossover, hoping to trigger some useful concept formation."

As noted above, this component of CogPrime views the whole AtomSpace as a kind of genetic algorithm - but the fitness function is "ecological" rather than fixed, and of course the crossover and mutation operators are highly specialized. Most of the concepts produced through evolutionary operations are going to be useless nonsense, but will be recognized by the importance updating process and subsequently forgotten from the system. The useful ones will link into other concepts and become ongoing aspects of the system's mind. The importance updating process amounts to fitness evaluation, and it depends implicitly on the sum total of the cognitive processes going on in CogPrime .

To ensure that importance updating properly functions as fitness evaluation, it is critical that evolutionarily-created concepts (and other speculatively created Atoms) always comprise a small percentage of the total concepts in the system. This guarantees that importance will serve as a meaningful "fitness function" for newly created ConceptNodes. The reason for this is that the importance measures how useful the newly created node is, in the context of the previously existing Atoms. If there are too many speculative, possibly useless new ConceptNodes in the system at once, the importance becomes an extremely noisy fitness measure, as it's largely measuring the degree to which instances of new nonsense fit in with other instances of new nonsense. One may find interesting self-organizing phenomena in this way, but in an AGI context we are not interested in undirected spontaneous pattern-formation, but rather in harnessing self-organizing phenomena toward system goals. And the latter is achieved by having a modest but not overwhelming amount of speculative new nodes entering into the system.

Finally, as discussed earlier, evolutionary operations on maps may occur naturally and automatically as a consequence of other cognitive operations.

Maps are continually mutated due to fluctuations in system dynamics; and maps may combine with other maps with which they overlap, as a consequence of the nonlinear properties of activation spreading and importance updating. Map-level evolutionary operations are not closely tied to their Atom-level counterparts (a difference from e.g. the close correspondence between map-level logical operations and underlying Atom-level logical operations).

38.3 Conceptual Blending

The notion of Conceptual Blending (aka Conceptual Integration) was proposed by Gilles Fauconnier and Mark Turner [FT02] as general theory of cognition. According to this theory, the basic operation of creative thought is the "blend" in which elements and relationships from diverse scenarios are merged together in a judicious way. As a very simple example, we may consider the blend of "tower" and "snake" to form a new concept of "snake tower" (a tower that looks somewhat like a snake). However, most examples of blends will not be nearly so obvious. For instance, the complex numbers could be considered a blend between 2D points and real numbers. Figure 38.1 gives a conceptual illustration of the blending process.

The production of a blend is generally considered to have three key stages (elucidated via the example of building a snake-tower out of blocks):

- *composition*: combining judiciously chosen elements from two or more concept inputs
  - *Example*: Taking the "buildingness" and "verticalness" of a tower, and the "head" and "mouth" and "tail" of a snake
- *completion*: adding new elements from implicit background knowledge about the concept inputs
  - *Example*: Perhaps a mongoose-building will be built out of blocks, poised in a position indicating it is chasing the snake-tower (incorporating the background knowledge that mongeese often chase snakes)
- *elaboration*: fine-tuning, which shapes the elements into a new concept, guided by the desire to optimize certain criteria
  - *Example*: The tail of the snake-tower is a part of the building that rests on the ground, and connects to the main tower. The head of the snake-tower is a portion that sits atop the main tower, analogous to the restaurant atop the Space Needle.

The "judiciousness" in the composition phase may be partially captured in CogPrime via PLN inference, via introducing a "consistency criterion" that the elements chosen as part of the blend should not dramatically decrease

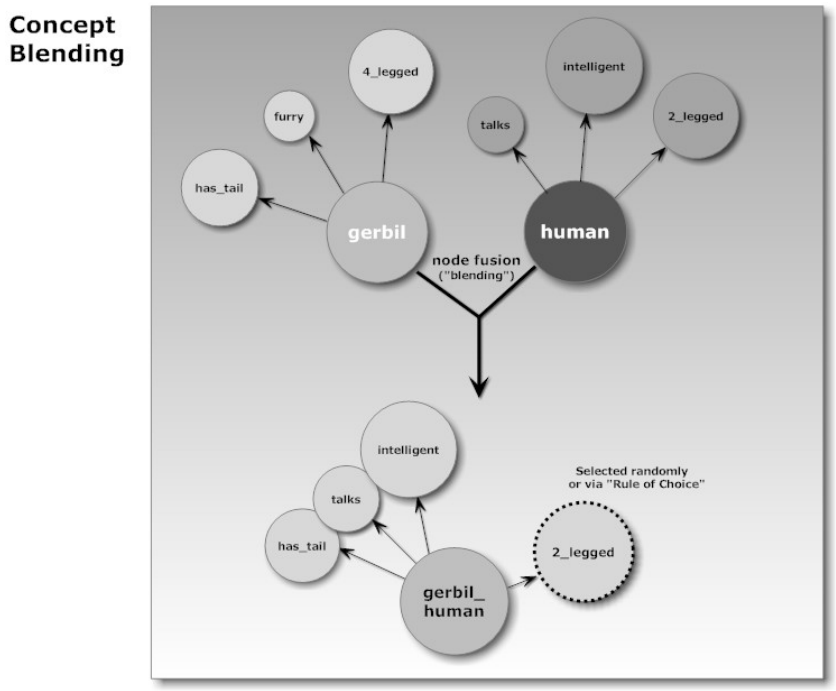


Fig. 38.1 Conceptual Illustration of Conceptual Blending

in confidence after the blend's relationships are submitted to PLN inference. One especially doesn't want to choose mutually contradictory elements from the two inputs. For instance one doesn't want to choose "alive" as an element of "snake", and "non-living" as an element of "building." This kind of contradictory choice can be ruled out by inference, because after very few inference steps, this choice would lead to a drastic confidence reduction for the InheritanceLinks to both "alive" and "non-living."

Aside from consistency, some other criteria considered relevant to evaluating the quality of a blend, are:

- *topology principle*: relations in the blend should match the relations of their counterparts in other concepts related to the concept inputs
- *web principle* that the representation in the blended space should maintain mappings to the concept inputs
- *unpacking principle* that, given a blended concept, the interpreter should be able to infer things about other related concepts
- *good reason principle* that there should be simple explanations for the elements of the blend
- *metonymic tightening* that when metonymically related elements are projected into the blended space, there is pressure to compress the "distance" between them.

While vague-sounding in their verbal formulations, these criteria have been computationally implemented in the Sapper system, which uses blending theory to model analogy and metaphor [VK94, VO07]; and in a different form in [Car06]'s framework for computational creativity. In CogPrime terms, these various criteria essentially boil down to: **the new, blended concept should get a lot of interesting links**.

One could implement blending in CogPrime very straightforwardly via an evolutionary approach: search the space of possible blends, evaluating each one according to its consistency but also the STI that it achieves when released into the Atomspace. However, this will be quite computationally expensive, so a wiser approach is to introduce heuristics aimed at increasing the odds of producing important blends.

A simple heuristic is to calculate, for each candidate blend, the amount of STI that the blend would possess N cycles later if, at the current time, it was given a certain amount of STI. A blend that would accumulate more STI in this manner may be considered more promising, because this means that its components are more richly interconnected. Further, this heuristic may be used as a guide for greedy heuristics for creating blends: e.g. if one has chosen a certain element A of the first blend input, then one may seek an element B of the second blend input that has a strong Hebbian link to A (if such a B exists).

However, it may also be interesting to pursue different sorts of heuristics, using information-theoretic or other mathematical criteria to preliminarily filter possible blends before they are evaluated more carefully via integrated cognition and importance dynamics.

38.3.1 Outline of a CogPrime Blending Algorithm

A rough outline of a concept blending algorithm for CogPrime is as follows:

- Choose a pair of concepts C1 and C2, which have a nontrivially-strong HebbianLink between them, but not an extremely high-strength SimilarityLink between them [i.e. the concepts should have something to do with each other, but not be extremely similar; blends of extremely similar things are boring]. These parameters may be twiddled.
- Form a new concept C3, which has some of C1's links, and some of C2's links
- If C3 has obvious contradictions, resolve them by pruning links. (For instance, if C1 inherits from alive to degree .9 and C2 inherits from alive to degree .1, then one of these two TruthValue versions for the inheritance link from alive, has got to be pruned...)
- For each of C3's remaining links L, make a vector indicating everything it or its targets are associated with (via HebbianLinks or other links). This

is basically a list of "what's related to L". Then, assess whether there are a lot of common associations to the links L that came from C1 and the links L that came from C2
- If the filter in step 4 is passed, then let the PLN forward chainer derive some conclusions about C3, and see if it comes up with anything interesting (e.g. anything with surprising truth value, or anything getting high STI, etc.)

Steps 1 and 2 should be repeated over and over. Step 5 is basically "cognition as usual" – i.e. by the time the blended concept is thrown into the Atomspace and subjected to Step 5, it's being treated the same as any other ConceptNode.

The above is more of a meta-algorithm than a precise algorithm. Many avenues for variation exist, including

- Step 1: heuristics for choosing what to try to blend
- Step 3: how far do we go here, at removing contradictions? Do we try simple PLN inference to see if contradictions are unveiled, or do we just limit the contradiction-check to seeing if the same exact link is given different truth-values?
- Step 4: there are many different ways to build this association-vector. There are also many ways to measure whether a set of association-vectors demonstrates "common associations". Interaction information [**?**] is one fancy way; there are also simpler ones.
- Step 5: there are various ways to measure whether PLN has come up with anything interesting

38.3.2 Another Example of Blending

To illustrate these ideas further, consider the example of the SUV – a blend of "Car" and "Jeep"

Among the relevant properties of Car are:

- appealing to ordinary consumers
- fuel efficient
- fits in most parking spots
- easy to drive
- 2 wheel drive

Among the relevant properties of Jeep are:

- 4 wheel drive
- rugged
- capable of driving off road
- high clearance

- open or soft top

Obviously, if we want to blend Car and Jeep, we need to choose properties of each that don't contradict each other. We can't give the Car/Jeep both 2 wheel drive and 4 wheel drive. 4 wheel drive wins for Car/Jeep because sacrificing it would get rid of "capable of driving off road", which is critical to Jeep-ness; whereas sacrificing 2WD doesn't kill anything that's really critical to car-ness.

On the other hand, having a soft top would really harm "appealing to consumers", which from the view of car-makers is a big part of being a successful car. But getting rid of the hard top doesn't really harm other aspects of jeep-ness in any series way.

However, what really made the SUV successful was that "rugged" and "high clearance" turned out to make SUVs look funky to consumers, thus fulfilling the "appealing to ordinary consumers" feature of Car. In other words, the presence of the links

- looks funky → appealing to ordinary consumers
- rugged & high clearance → looks funky

made a big difference. This is the sort of thing that gets figured out once one starts doing PLN inference on the links associated with a candidate blend.

However, if one views each feature of the blend as a probability distribution over concept space (e.g., indicating how closely associated with feature is with each other concept (e.g. via HebbianLinks) É then we see that the mutual information (and more generally interaction information) between the features of the blend, is a quick estimate of how likely it is that inference will lead to interesting conclusions via reasoning about the combination of features that the blend possesses.

38.4 Clustering

Next, a different method for creating new ConceptNodes in CogPrime is using clustering algorithms. There are many different clustering algorithms in the statistics and data mining literature, and no doubt many of them could have value inside CogPrime . We have experimented with several different clustering algorithms in the CogPrime context, and have selected one, which we call Omniclust [GCPM06], based on its generally robust performance on high-volume, *noisy* data. However, other methods such as EM (Expectation-Maximization) clustering [WF05] would likely serve the purpose very well also.

In the above discussion on evolutionary concept creation, we mentioned the use of a clustering algorithm to cluster links. The same algorithm we describe here for clustering ConceptNodes directly and creating new ConceptNodes

representing these clusters, can also be used for clustering links in the context of node mutation and crossover.

The application of Omniclust or any other clustering algorithm for ConceptNode creation in CogPrime is simple. The clustering algorithm is run periodically, and the most significant clusters that it finds are embodied as ConceptNodes, with InheritanceLinks to their members. If these significant clusters have subclusters also identified by Omniclust, then these subclusters are also made into ConceptNodes, etc., with InheritanceLinks between clusters and subclusters.

Clustering technology is famously unreliable, but this unreliability may be mitigated somewhat by using clusters as initial guesses at concepts, and using other methods to refine the clusters into more useful concepts. For instance, a cluster may be interpreted as a disjunctive predicate, and a search may be made to determine sub-disjunctions about which interesting PLN conclusions may be drawn.

38.5 Concept Formation via Formal Concept Analysis

Another approach to concept formation is an uncertain version of Formal Concept Analysis [GSW05]. There are many ways to create such a version, here we describe one approach we have found interesting, called Fuzzy Concept Formation (FCF).

The general formulation of FCF begins with n objects $O_1, ..., O_n$, m basic attributes $a_1, ..., a_m$, and information that object O_i possesses attribute a_j to degree $w_{ij} \in [0, 1]$. In CogPrime , the objects and attributes are Atoms, and w_{ij} is the strength of the InheritanceLink pointing from O_i to a_j.

In this context, we may define a **concept** as a fuzzy set of objects, and a **derived attribute** as a fuzzy set of attributes.

Fuzzy concept formation (FCF) is, then, a process that produces N "concepts" $C_{n+1}, ..., C_{n+N}$ and M "derived attributes" $d_{m+1}, ..., d_{m+M}$, based on the initial set of objects and attributes. We can extend the weight matrix w_{ij} to include entries involving concepts and derived attributes as well, so that e.g. $w_{n+3,m+5}$ indicates the degree to which concept C_{n+3} possesses derived attribute d_{m+5}.

The learning engine underlying FCF is a clustering algorithm $clust = clust(X_1, ..., X_r; b)$ which takes in r vectors $X_r \in [0, 1]^n$ and outputs b or fewer clusters of these vectors. The overall FCF process is independent of the particular clustering algorithm involved, though the interestingness of the concepts and attributes formed will of course vary widely based on the specific clustering algorithm. Some clustering algorithms will work better with large values of b, others with smaller values of b.

We then define the process $form\_concepts(b)$ to operate as follows. Given a set $S = S_1, ..., S_k$ containing objects, concepts, or a combination of objects

and concepts, and an attribute vector w_i of length h with entries in $[0, 1]$ corresponding to each S_i, one applies *clust* to find b clusters of attribute vectors w_i: $B_1, ..., B_b$. Each of these clusters may be considered as a fuzzy set, for instance by considering the membership of x in cluster B to be $2^{-d(x,centroid(B))}$ for an appropriate metric d. These fuzzy sets are the b concepts produced by $form\_concepts(b)$.

38.5.1 Calculating Membership Degrees of New Concepts

The degree to which a concept defined in this way possesses an attribute, may be defined in a number of ways, maybe the simplest is: weighted-summing the degree to which the members of the concept possess the attribute. For instance, to figure out the degree to which beautiful women (a concept) are isane (an attribute), one would calculate

$$\frac{\sum_{w \in beautiful\_women} \chi_{beautiful\_women}(w) \chi_{insane}(w)}{\sum_{w \in beautiful\_women} \chi_{beautiful\_women}(w)}$$

where $\chi_X(w)$ denotes the fuzzy membership degree of w in X. One could probably also consider *ExtensionalInheritance beautiful_ women insane*.

38.5.2 Forming New Attributes

One may define an analogous process $form\_attributes(b)$ that begins with a set $A = A_1, ..., A_k$ containing (basic and/or derived) attributes, and a column vector

$$\begin{pmatrix} w_{1i} \\ ... \\ w_{hi} \end{pmatrix}$$

of length h with entries in $[0, 1]$ corresponding to each A_i (the column vector tells the degrees to which various objects possess the attributes A_i). One applies *clust* to find b clusters of vectors v_i: $B_1, ..., B_b$. These clusters may be interpreted as fuzzy sets, which are derived attributes.

38.5.2.1 Calculating Membership Degrees of New, Derived Attributes

One must then defines the degree to which an object or concept possesses a derived attribute. One way to do this is using a geometric mean. For instance, suppose there is a derived attribute formed by combining the attributes *vain*, *selfish* and *egocentric*. Then, the degree to which the concept *banker* possesses this new derived attribute could be defined by

$$\frac{\sum_{b \in banker} \chi_{banker}(b) \left(\chi_{vain}(b) \chi_{selfish}(b) \chi_{egocentric}(b)\right)^{1/3}}{\sum_{b \in banker} \chi_{banker}(b)}$$

38.5.3 Iterating the Fuzzy Concept Formation Process

Given a set S of concepts and/or objects with a set A of attributes, one may define

- $append\_concepts(S', S)$ as the result of adding the concepts in the set S' to S, and evaluating all the attributes in A on these concepts, to get an expanded matrix w
- $append\_attributes(A', A)$ as the result of adding the attributes in the set A' to A, and evaluating all the attributes in A' on the concepts and objects in S, to get an expanded matrix w
- $collapse(S, A)$ is the result of taking (S, A) and eliminating any concept or attribute that has distance less than ϵ from some other concept or attribute that comes before it in the lexicographic ordering of concepts or attributes. I.e., *collapse* removes near-duplicate concepts or attributes.

Now, one may begin with a set S of objects and attributes, and iteratively run a process such as

```
b = r^c \\e.g. r=2, or r=1.5
while(b>1) {
   S = append_concepts(S, form_concepts(S,b))
   S = collapse(S)
   S = append_attributes(S, form_attributes(S,b))
   S = collapse(S)
   b = b/r
}
```

with c corresponding to the number of iterations. This will terminate in finite time with a finitely expanded matrix w containing a number of concepts and derived attributes in addition to the original objects and basic attributes.

Or, one may look at

```
while(S is different from old_S) {
   old_S = S
   S = add_concepts(S, form_concepts(S,b))
   S = collapse(S)
   S = add_attributes(S, form_attributes(S,b))
   S = collapse(S)
}
```

This second version raises the mathematical question of the speed with which it will terminate (as a function of ϵ). I.e., when does the concept and attribute formation process converge, and how fast? This will surely depend on the clustering algorithm involved.

Section XI
Integrative Learning

Chapter 39
Dimensional Embedding

39.1 Introduction

Among the many key features of the human brain omitted by typical formal neural network models, one of the foremost is the brain's three-dimensionality. The brain is not just a network of neurons arranged as an abstract graph; it's a network of neurons arranged in three-dimensional space, and making use of this three-dimensionality directly and indirectly in various ways and for various purposes. The somatosensory cortex contains a geometric map reflecting, approximatively, the geometric structure of parts of the body. Visual cortex uses the 2D layout of cortical sheets to reflect the geometric structure of perceived space; motion detection neurons often fire in the actual physical direction of motion, etc. The degree to which the brain uses 2D and 3D geometric structure to reflect conceptual rather than perceptual or motoric knowledge is unclear, but we suspect considerable. One well-known idea in this direction is the "self-organizing map" or Kohonen net [**?**], a highly effective computer science algorithm that performs automated classification and clustering via projecting higher-dimensional (perceptual, conceptual or motoric) vectors into a simulated 2D sheet of cortex.

It's not clear that the exploitation of low-dimensional geometric structure is something an AGI system necessarily *must* support – there are always many different approaches to any aspect of the AGI problem. However, the brain does make clear that exploitation of this sort of structure is a powerful way to integrate various useful heuristics. In the context of mind-world correspondence theory, there seems clear potential value in having a mind mirror the dimensional structure of the world, at some level of approximation.

It's also worth emphasizing that the brain's 3D structure has minuses as well as plusses – one suspects it complexities and constrains the brain, along with implicitly suggesting various useful heuristics. Any mathematical graph can be represented in 3 dimensions without links crossing (unlike in 2 dimensions), but that doesn't mean the representation will always be effi-

cient or convenient – sometimes it may result in conceptually related, and/or frequently interacting, entities being positioned far away from each other geometrically. Coupled with noisy signaling methods such as the brain uses, this sometime lack of alignment between conceptual/pragmatic and geometric structure can lead to various sorts of confusion (i.e. when neuron A sends a signal to physical distant neurons B, this may cause various side-effects along the path, some of which wouldn't happen if A and B were close to each other).

In the context of CogPrime , the most extreme way to incorporate a brain-like 3D structure would be to actually embed an Atomspace in a bounded 3D region. Then the Atomspace would be geometrically something like a brain, but with abstract nodes and links (some having explicit symbolic content) rather than purely sub symbolic neurons. This would not be a ridiculous thing to do, and could yield interesting results. However, we are unsure this would be an optimal approach. Instead we have opted for a more moderate approach: couple the non-dimensional Atomspace with a dimensional space, containing points corresponding to Atoms. That is, we perform an embedding of Atoms in the OpenCog AtomSpace into n-dimensional space – a judicious transformation of (hyper)graphs into vectors.

This embedding has applications to PLN inference control, and to the guidance of instance generation in PEPL learning of Combo trees. It is also, in itself, a valuable and interesting heuristic for sculpting the *link topology* of a CogPrime AtomSpace. The basic dimensional embedding algorithm described here is fairly simple and not original to CogPrime , but it has not previously been applied in any similar context.

The intuition underlying this approach is that there are some cases (e.g. PLN control, and PEPL guidance) where dimensional geometry provides a useful heuristic for constraining a huge search space, via providing a compact way of storing a large amount of information. Dimensionally embedding Atoms lets CogPrime be dimensional like the brain when it needs to be, yet with the freedom of nondimensionality the rest of the time. This dual strategy is one that may be of value for AGI generally beyond the CogPrime design, and is somewhat related to (though different in detail from) the way the CLARION cognitive architecture [SZ04] maps declarative knowledge into knowledge appropriate for its neural net layer.

There is an obvious way to project CogPrime Atoms into n-dimensional space, by assigning each Atom a numerical vector based on the weights of its links. But this is not a terribly useful approach, because the vectors obtained in this way will live, potentially, in millions- or billions-dimensional space. The approach we describe here is a bit different. We are defining more specific embeddings, each one based on a particular link type or set of link types. And we are doing the embedding into a space whose dimensionality is *high but not too high*, e.g. n=50. This moderate dimensional space could then be projected down into a lower dimensional space, like a 3D space, if needed.

The philosophy underlying the ideas proposed here is similar to that underlying Principal Components Analysis (PCA) in statistics [**?**]. The n-dimensional spaces we define here, like those used in PCA or LSI (for Latent Semantic Indexing [LMDK07]), are defined by sets of *orthogonal concepts* extracted from the original space of concepts. The difference is that PCA and LSI work on spaces of entities defined by feature vectors, whereas the methods described here work for entities defined as nodes in weighted graphs. There is no precise notion of *orthogonality* for nodes in a weighted graph, but one can introduce a reasonable proxy.

39.2 Link Based Dimensional Embedding

In this section we define the type of dimensional embedding that we will be talking about. For concreteness we will speak in terms of CogPrime nodes and links but the discussion applies much more generally than that.

A *link based dimensional embedding* is defined as a mapping that maps a set of CogPrime Atoms into points in an n-dimensional real space, by:

- mapping link strength into coordinate values in an embedding space, and
- representing nodes as points in this embedding space, using the coordinate values defined by the strengths of their links.

In the usual case, a dimensional embedding is formed from links of a single type, or from links whose types are very closely related (e.g. from all symmetrical logical links).

Mapping all the link strengths of the links of a given type into coordinate values in a dimensional space is a simple, but not a very effective strategy. The approach described here is based on strategically choosing a subset of particular links and forming coordinate values from them. The choice of links is based on the desire for a correspondence between the metric structure of the embedding space, and the metric structure implicit in the weights of the links of the type being embedded. The basic idea of metric preservation is depicted in Figure 39.1.

More formally, let $proj(A)$ denote the point in R^n corresponding to the Atom A. Then if, for example, we are doing an embedding based on SimilarityLinks, we want there to be a strong correlation (or rather anticorrelation) between:

```
(SimilarityLink A B).tv.s
```

and

$$d_E(proj(A), proj(B))$$

where d_E denotes the Euclidean distance on the embedding space. This is a simple case because SimilarityLink is symmetric. Dealing with asymmetric

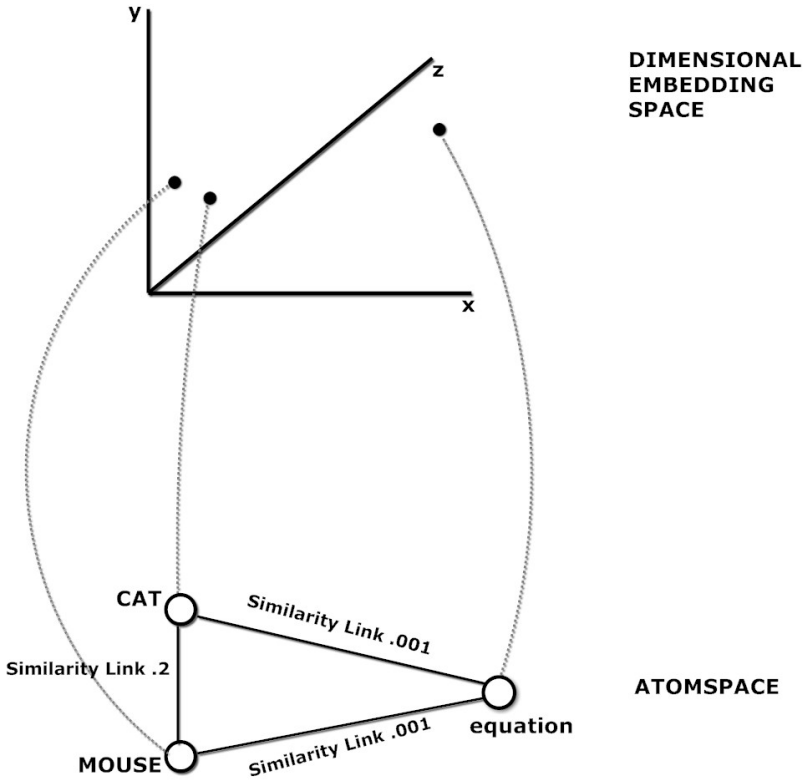


Fig. 39.1 Metric-Preserving Dimensional Embedding. The basic idea of the sort of embedding described here is to map Atoms into numerical vectors, in such a way that, on average, distance between Atoms roughly correlates with distance between corresponding vectors. (The picture shows a 3D embedding space for convenience, but in reality the dimension of the embedding space will generally be much higher.)

links like InheritanceLinks is a little subtler, and will be done below in the context of inference control.

Larger dimensions generally allow greater correlation, but add complexity. If one chooses the dimensionality equal to the number of nodes in the graph, there is really no point in doing the embedding. On the other hand, if one tries to project a huge and complex graph into 1 or 2 dimensions, one is bound to lose a lot of important structure. The optimally useful embedding will be into a space whose dimension is *large but not too large.*

For internal CogPrime inference purposes, we should generally use a moderately high-dimensional embedding space, say n=50 or n=100.

39.3 Harel and Koren's Dimensional Embedding Algorithm

Our technique for embedding CogPrime Atoms into high-dimensional space is based on an algorithm suggested by David Harel and Yehuda Koren [HK02]. Their work is concerned with visualizing large graphs, and they propose a two-phase approach:

1. embed the graph into a high-dimensional real space
2. project the high-dimensional points into 2D or 3D space for visualization

In CogPrime , we don't always require the projection step (step 2); our focus is on the initial embedding step. Harel and Koren's algorithm for dimensional embedding (step 1) is directly applicable to the CogPrime context.

Of course this is not the only embedding algorithm that would be reasonable to use in an CogPrime context; it's just one possibility that seems to make sense.

Their algorithm works as follows.

Suppose one has a graph with symmetric weighted links. Further, assume that between any two nodes in the graph, there is a way to compute the weight that a link between those two nodes would have, even if the graph in fact doesn't contain a link between the two nodes.

In the CogPrime context, for instance, the nodes of the graph may be ConceptNodes, and the links may be SimilarityLinks. We will discuss the extension of the approach to deal with asymmetric links like InheritanceLinks, later on.

Let n denote the dimension of the embedding space (e.g. $n = 50$). We wish to map graph nodes into points in R^n, in such a way that the weight of the graph link between A and B correlates with the distance between $proj(A)$ and $proj(B)$ in R^n.

39.3.1 Step 1: Choosing Pivot Points

Choose n "pivot points" that are roughly uniformly distributed across the graph.

To do this, one chooses the first pivot point at random and then iteratively chooses the i'th point to be maximally distant from the previous $(i-1)$ points chosen.

One may also use additional criteria to govern the selection of pivot points. In CogPrime , for instance, we may use *long-term stability* as a secondary criterion for selecting Atoms to serve as pivot points. Greater computational efficiency is achieved if the pivot-point Atoms don't change frequently.

39.3.2 Step 2: Similarity Estimation

Estimate the similarity between each Atom being projected, and the n pivot Atoms.

This is expensive. However, the cost is decreased somewhat in the CogPrime case by caching the similarity values produced in a special table (they may not be important enough otherwise to be preserved in CogPrime). Then, in cases where neither the pivot Atom nor the Atom being compared to it have changed recently, the cached value can be reused.

39.3.3 Step 3: Embedding

Create an n-dimensional space by assigning a coordinate axis to each pivot Atom. Then, for an Atom i, the i'th coordinate value is given by its similarity to the i'th pivot Atom.

After this step, one has transformed one's graph into a collection of n-dimensional vectors. WIKISOURCE:EmbeddingBasedInference

39.4 Embedding Based Inference Control

One important application for dimensional embedding in CogPrime is to help with the control of

- Logical inference
- Direct evaluation of logical links

We describe how it can be used specifically to stop the CogPrime system from continually trying to make the same unproductive inferences.

To understand the problem being addressed, suppose the system tries to evaluate the strength of the relationship

```
SimilarityLink foot toilet
```

Assume that no link exists in the system representing this relationship.

Here "foot" and "toilet" are hypothetical ConceptNodes that represent aspects of the concepts of foot and toilet respectively. In reality these concepts might well be represented by complex maps rather than individual nodes.

Suppose the system determines that the strength of this Link is very close to zero. Then (depending on a threshold in the MindAgent), it will probably not create a SimilarityLink between the "foot" and "toilet" nodes.

Now, suppose that a few cycles later, the system again tries to evaluate the strength of the same Link,

```
SimilarityLink foot toilet
```

Again, very likely, it will find a low strength and not create the Link at all.

The same problem may occur with InheritanceLinks, or any other (first or higher order) logical link type.

Why would the system try, over and over again, to evaluate the strength of the same nonexistent relationship? Because the control strategies of the current forward-chaining inference and pattern mining MindAgents are simple by design. These MindAgents work by selecting Atoms from the AtomTable with probability proportional to importance, and trying to build links between them. If the foot and toilet nodes are both important at the same time, then these MindAgents will try to build links between them - regardless of how many times they've tried to build links between these two nodes in the past and failed.

How do we solve this problem using dimensional embedding? Generally:

- one will need a different embedding space for each link type for which one wants to prevent repeated attempted inference of useless relationships. Sometimes, very closely related link types might share the same embedding space; this must be decided on a case-by-case basis.
- in the embedding space for a link type L, one only embeds Atoms of a type that can be related by links of type L

It is too expensive to create a new embedding very often. Fortunately, when a new Atom is created or an old Atom is significantly modified, it's easy to reposition the Atom in the embedding space by computing its relationship to the pivot Atoms. Once enough change has happened, however, new pivot Atoms will need to be recomputed, which is a substantial computational expense. We must update the pivot point set every N cycles, where N is large; or else, whenever the total amount of change in the system has exceeded a certain threshold.

Now, how is this embedding used for inference control? Let's consider the case of similarity first. Quite simply, one selects a pair of Atoms (A,B) for SimilarityMining (or inference of a SimilarityLink) based on some criterion such as, for instance:

```
importance(A) * importance(B) * simproj(A,B)
```

where

```
distproj(A,B) = dE( proj(A) , proj(B) )

simproj = 2-c*distproj
```

and c is an important tunable parameter.

What this means is that, if A and B are far apart in the SimilarityLink embedding space, the system is unlikely to try to assess their similarity.

There is a tremendous space efficiency of this approach, in that, where there are N Atoms and m pivot Atoms, N^2 similarity relationships are being

approximately stored in m*N coordinate values. Furthermore, the cost of computation is m*N times the cost of assessing a single SimilarityLink. By accepting crude approximations of actual similarity values, one gets away with linear time and space cost.

Because this is just an approximation technique, there are definitely going to be cases where A and B are not similar, even though they're close together in the embedding space. When such a case is found, it may be useful for the AtomSpace to explicitly contain a low-strength SimilarityLink between A and B. This link will prevent the system from making false embedding-based decisions to explore (SimilarityLink A B) in the future. Putting explicit low-strength SimilarityLinks in the system in these cases, is obviously much cheaper than using them for all cases.

We've been talking about SimilarityLinks, but the approach is more broadly applicable. Any symmetric link type can be dealt with about the same way. For instance, it might be useful to keep dimensional embedding maps for

- SimilarityLink
- ExtensionalSimilarityLink
- EquivalenceLink
- ExtensionalEquivalenceLink

On the other hand, dealing with asymmetric links in terms of dimensional embedding requires more subtlety – we turn to this topic below.

39.5 Dimensional Embedding and InheritanceLinks

Next, how can we use dimensional embedding to keep an approximate record of which links do not inherit from each other? Because inheritance is an asymmetric relationship, whereas distance in embedding spaces is a symmetrical relationship, there's no direct and simple way to do so.

However, there is an indirect approach that solves the problem, which involves maintaining two embedding spaces, and combining information about them in an appropriate way. In this subsection, we'll discuss an approach that should work for InheritanceLink, SubsetLink, ImplicationLink, and ExtensionalImplicationLink and other related link types. But we'll explicitly present it only for the InheritanceLink case.

Although the embedding algorithm described above was intended for symmetric weighted graphs, in fact we can use it for asymmetric links in just about the same way. The use of the embedding graph for inference control differs, but not the basic method of defining the embedding.

In the InheritanceLink case, we can define pivot Atoms in the same way, and then we can define two vectors for each Atom A:

```
proj_{parent}(A)_i = (InheritanceLink A A_i).tv.s
```

```
proj_{child}(A)_i = (InheritanceLink A_i A).tv.s
```

where A_i is the i'th pivot Atom.

If generally $proj_{child}(A)_i \leq proj_{child}(B)_i$ then qualitatively "children of A are children of B"; and if generally $proj_{parent}(A)_i \geq proj_{paret}(B)_i$ then qualitatively "parents of B are parents of A". The combination of these two conditions means heuristically that $(Inheritance\ A\ B)$ is likely. So, by combining the two embedding vectors assigned to each Atom, one can get heuristic guidance regarding inheritance relations, analogous to the case with similarity relationships. One may produce mathematical formulas estimating the error of this approach under appropriate conditions, but in practice it will depend on the probability distribution of the vectors.

Chapter 40
Mental Simulation and Episodic memory

40.1 Introduction

This brief chapter deals with two important, coupled cognitive components of CogPrime : the component concerned with creating *internal simulations* of situations and episodes in the external physical world, and the one concerned with storing and retrieving memories of situations and episodes.

This are components that are likely significantly different in CogPrime from anything that exists in the human brain, yet, the functions they carry out are obviously essential to human cognition (perhaps more so to human cognition than to CogPrime 's cognition, because CogPrime is by design more reliant on formal reasoning than the human brain is).

Much of human thought consists of internal, quasi-sensory "imaging" of the external physical world – and much of human memory takes place of remembering autobiographical situations and episodes from daily life, or from stories heard from others or absorbed via media. Often this episodic remembering takes the form of visualization, but not always. Blind people generally think and remember in terms of non-visual imagery, and many sighted people think in terms of sounds, tastes or smells in addition to visual images.

So far, the various mechanisms proposed as part of CogPrime do not have much to do with either internal imagery or episodic remembering, even though both seem to play a large role in human thought. This is OK, of course, since CogPrime is not intended as a simulacrum of human thought, but rather as a different sort of intelligence.

However, we believe it will actually be valuable to CogPrime to incorporate both of these factors. And for that purpose, we propose

- a novel mechanism: the incorporation within the CogPrime system of a 3D physical-world simulation engine.
- an episodic memory store centrally founded on dimensional embedding, and linked to the internal simulation model

40.2 Internal Simulations

The current use of virtual worlds for OpenCog is to provide a space in which human-controlled agents and CogPrime -controlled agents can interact, thus allowing flexible instruction of the CogPrime system by humans, and flexible embodied, grounded learning by CogPrime systems. But this very same mechanism may be used internally to CogPrime , i.e. a CogPrime system may be given an *internal simulation world*, which serves as a sort of "mind's eye." Any sufficiently flexible virtual world software may be used for this purpose, for example OpenSim (http:\opensim.org).

Atoms encoding percepts may be drawn from memory and used to generate forms within the internal simulation world. These forms may then interact according to

- the patterns via which they are remembered to act
- the laws of physics, as embodied in the simulation world

This allows a kind of "implicit memory," in that patterns emergent from the world-embedded interaction of a number of entities *need not explicitly be stored in memory*, so long as they will emerge when the entities are reawakened within the internal simulation world.

The SimulatorMindAgent grabs important perceptual Atoms and uses them to generate forms within the internal simulation world, which then act according to remembered dynamical patterns, with the laws of physics filling in the gaps in memory. This provides a sort of running internal visualization of the world. Just as important, however, are specific schemata that utilize visualization in appropriate contexts. For instance, if reasoning is having trouble solving a problem related to physical entities, it may feed these entities to the internal simulation world to see what can be discovered. Patterns discovered via simulation can then be fed into reasoning for further analysis.

The process of perceiving events and objects in the simulation world is essentially identical to the process of perceiving events and objects in the "actual" world.

And of course, an internal simulation world may be used whether the CogPrime system in question is hooked up to a virtual world like OpenSim, or to a physical robot.

Finally, perhaps the most interesting aspect of internal simulation is the generation of "virtual perceptions" from abstract concepts. Analogical reasoning may be used to generate virtual perceptions that were never actually perceived, and these may then be visualized. The need for "reality discrimination" comes up here, and is easier to enforce in CogPrime than in humans. A PerceptNode that was never actually perceived may be explicitly embedded in a HypotheticalLink, thus avoiding the possibility of confusing virtual percepts with actual ones. How useful the visualization of virtual perceptions will be to CogPrime cognition, remains to be seen. This kind of visualization

is key to human imagination but this doesn't mean it will play the same role in CogPrime 's quite different cognitive processes. But it is important that CogPrime has the power to carry out this kind of imagination.

40.3 Episodic Memory

Episodic memory refers to the memory of our own "life history" that each of us has. Loss of this kind of memory is the most common type of amnesia in fiction – such amnesia is particularly dramatic because our episodic memories constitute so much of what we consider as our "selves." To a significant extent, we as humans remember, reason and relate in terms of *stories* – and the centerpiece of our understanding of stories is our episodic memory. A CogPrime system need not be as heavily story-focused as a typical human being (though it could be, potentially) – but even so, episodic memory is a critical component of any CogPrime system controlling an agent in a world.

The core idea underlying CogPrime 's treatment of episodic memory is a simple one: two dimensional embedding spaces dedicated to episodes. An episode – a coherent collection of happenings, often with causal interrelationships, often (but not always) occurring near the same spatial or temporal locations as each other – may be represented explicitly as an Atom, and implicitly as a map whose key is that Atom. These episode-Atoms may then be mapped into two dedicated embedding spaces:

- one based on a distance metric determined by spatiotemporal proximity
- one based on a distance metric determined by semantic similarity

A *story* is then a series of episodes – ideally one that, if the episodes in the series become important sequentially in the AtomSpace, causes a significant emotional response in the system. Stories may also be represented as Atoms, in the simplest case consisting of SequentialAND links joining episode-Atoms. Stories then correspond to paths through the two episodic embedding spaces. Each path through each embedding space implicitly has a sort of "halo" in the space – visualizable as a tube snaking through the space, centered on the path. This tube contains other paths – other stories – that related to the given center story, either spatiotemporally or semantically.

The familiar everyday human experience of episodic memory may then be approximatively emulated via the properties of the dimensional embedding space. For instance, episodic memory is famously associative – when we think of one episode or story, we think of others that are spatiotemporally or semantically associated with it. This emerges naturally from the embedding space approach, due to the natural emergence of distance-based associative memory in an embedding space.

Figures 40.1 and 40.2 roughly illustrates the link between episodic/perceptual and declarative memory.

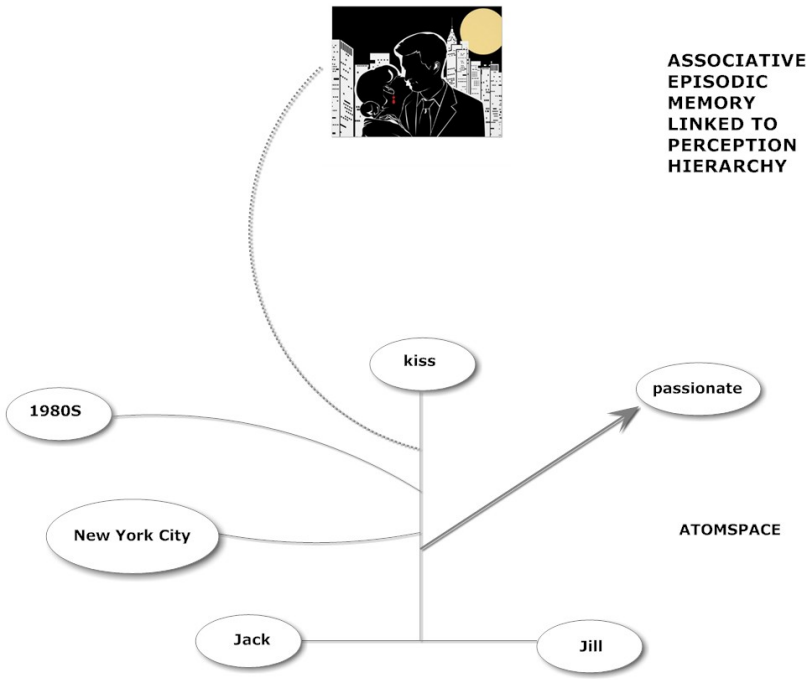


Fig. 40.1 Relationship Between Episodic, Declarative and Perceptual Memory. The nodes and links at the bottom depict declarative memory stored in the Atomspace; the picture at the top illustrates an archetypal episode stored in episodic memory, and linked to the perceptual hierarchy enabling imagistic simulation.

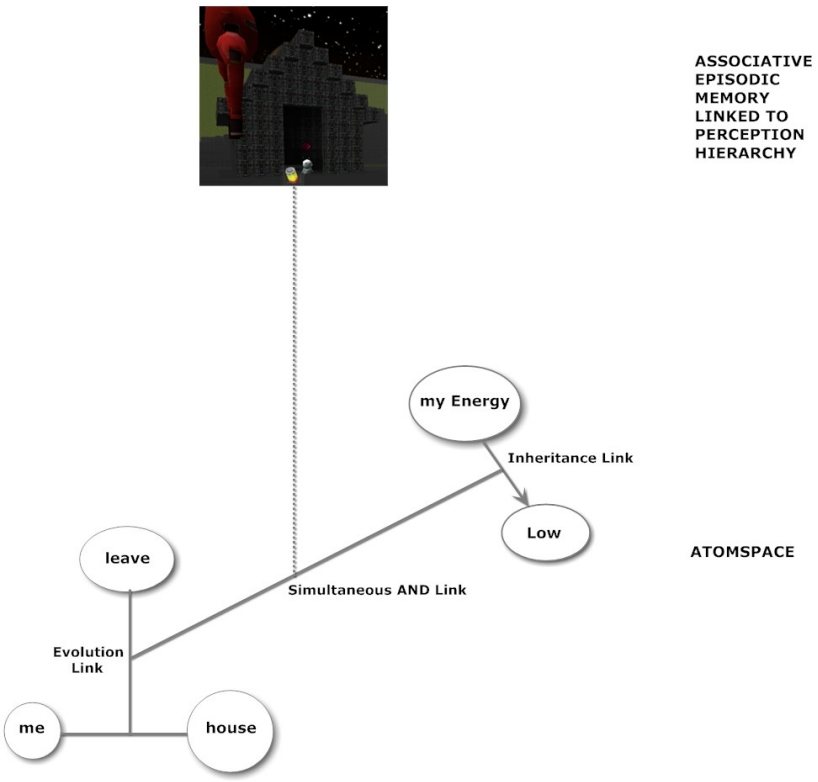


Fig. 40.2 Relationship Between Episodic, Declarative and Perceptual Memory. Another example similar to the one in **??**, but referring specifically to events occurring in an OpenCogPrime -controlled agent's virtual world.

Chapter 41
Integrative Procedure Learning

41.1 Introduction

"Procedure learning" – learning step-by-step procedures for carrying out internal or external operations – is a highly critical aspect of general intelligence, and is carried out in CogPrime via a complex combination of methods. This somewhat heterogeneous chapter reviews several advanced aspects of procedure learning in CogPrime , mainly having to do with the integration between different cognitive processes.

In terms of general cognitive theory and mind-world correspondence, this is some of the subtlest material in the book. We are not concerned just with how the mind's learning of one sort of knowledge correlated with the way this sort of knowledge is structured in the mind's habitual environments, in the context of its habitual goals. Rather, we are concerned with how various sorts of knowledge intersect and interact with each other. The proposed algorithmic intersections between, for instance, declarative and procedural learning processes, are reflective of implicit assumptions about how declarative and procedural knowledge are presented in the world in the context of the system's goals – but these implicit assumptions are not always easy to tease out and state in a compact way. We will do our best to highlight these assumptions as they arise throughout the chapter.

Key among these assumptions, however, are that a human-like mind

- is presented with various procedure learning problems at various levels of difficulty (so that different algorithms may be appropriate depending on the difficulty level). *This leads for instance to the possibility of using various different algorithms like MOSES or hill climbing, for different procedure learning problems.*
- is presented with some procedure learning problems that may be handled in a relatively isolated way, and others that are extremely heavily dependent on context, often in a way that recurs across multiple learning instances in similar contexts. *This leads to a situations where the value of*

bringing declarative (PLN) and associative (ECAN) and episodic knowledge into the procedure learning process, has varying value depending on the situation.

- is presented with a rich variety of procedure learning problems with complex interrelationships, including many problems that are closely related to previously solved problems in various ways. *This highlights the value of using PLN analogical reasoning, and importance spreading along HebbianLinks learned by ECAN, to help guide procedure learning in various ways.*
- needs to learn some procedures whose execution may be carried out in a relatively isolated way, and other procedures whose execution requires intensive ongoing interaction with other cognitive processes

The diversity of procedure learning situations reflected in these assumptions, leads naturally to the diversity of technical procedure learning approaches described in this chapter. Potentially one could have a single, unified algorithm covering all the different sorts of procedure learning, but instead we have found it more practical to articulate a small number of algorithms which are then combined in different ways to yield the different kinds of procedure learning.

41.1.1 The Diverse Technicalities of Procedure Learning in CogPrime

On a technical level, this chapter discusses two closely related aspects of CogPrime : schema learning and predicate learning, which we group under the general category of "procedure learning."

Schema learning - the learning of SchemaNodes and schema maps (explained further in the Chapter 42) - is CogPrime lingo for learning how to do things. Learning how to act, how to perceive, and how to think - beyond what's explicitly encoded in the system's MindAgents. As an advanced CogPrime system becomes more profoundly self-modifying, schema learning will drive more and more of its evolution.

Predicate learning, on the other hand, is the most abstract and general manifestation of pattern recognition in the CogPrime system. PredicateNodes, along with predicate maps, are CogPrime 's way of representing general patterns (*general* within the constraints imposed by the system parameters, which in turn are governed by hardware constraints). Predicate evolution, predicate mining and higher-order inference - specialized and powerful forms of predicate learning - are the system's most powerful ways of creating general patterns in the world and in the mind. Simpler forms of predicate learning are grist for the mill of these processes.

It may be useful to draw an analogy with another (closely related) very hard problem in CogPrime , discussed in the PLN book: probabilistic logical unification, which in the CogPrime /PLN framework basically comes down to finding the SatisfyingSets of given predicates. Hard logical unification problems can often be avoided by breaking down large predicates into small ones in strategic ways, guided by non-inferential mind processes, and then doing unification only on the smaller predicates. Our limited experimental experience indicates that the same "hierarchical breakdown" strategy also works for schema and predicate learning, to an extent. But still, as with unification, even when one does break down a large schema or predicate learning problem into a set of smaller problems, one is still in most cases left with a set of fairly hard problems.

More concretely, CogPrime procedure learning may be generally decomposed into three aspects:

1. Converting back and forth between maps and ProcedureNodes (*encapsulation and expansion*)
2. Learning the Combo Trees to be embedded in grounded ProcedureNodes
3. Learning procedure maps (networks of grounded ProcedureNodes acting in a coordinated way to carry out procedures)

Each of these three aspects of CogPrime procedure learning mentioned above may be dealt with somewhat separately, though relying on largely overlapping methods.

CogPrime approaches these problems using a combination of techniques:

- Evolutionary procedure learning and hillclimbing for dealing with *brand new* procedure learning problems, requiring the origination of innovative, highly approximate solutions out of the blue
- Inferential procedure learning for taking approximate solutions and making them exact, and for dealing with procedure learning problems within domains where closely analogous procedure learning problems have previously been solved
- Heuristic, probabilistic data mining for the creation of encapsulated procedures (which then feed into inferential and evolutionary procedure learning), and the expansion of encapsulated procedures into procedure maps
- PredictiveImplicationLink formation (augmented by PLN inference on such links) as a CogPrime version of goal-directed reinforcement learning

Using these different learning methods together, as a coherently-tuned whole, one arrives at a holistic procedure learning approach that combines speculation, systematic inference, encapsulation and credit assignment in a single adaptive dynamic process.

We are relying on a combination of techniques to do what none of the techniques can accomplish on their own. The combination is far from arbi-

trary, however. As we will see, each of the techniques involved plays a unique and important role.

41.1.1.1 Comments on an Alternative Representational Approach

We briefly pause to contrast certain technical aspects of the present approach to analogous aspects of the Webmind AI Engine (one of CogPrime 's predecessor AI systems, briefly discussed above). This predecessor system used a knowledge representation somewhat similar to the Atomspace, but with various differences; for instance the base types were Node and Link rather than Atom, and there was a Node type not used in CogPrime called the SchemaInstanceNode (each one corresponding to a particular instance of a SchemaNode, used within a particular procedure).

In this approach, complex, learned schema were represented as distributed networks of elementary SchemaInstanceNodes, but these networks were not defined purely by function application - they involved explicit passing of variable values through VariableNodes. Special logic-gate-bearing objects were created to deal with the distinction between arguments of a SchemaInstanceNode, and *predecessor* tokens giving a SchemaInstanceNode permission to act.

While this approach is in principle workable, it proved highly complex in practice, and for the Novamente Cognition Engine and CogPrime we chose to store and manipulate procedural knowledge separately from declarative knowledge (via Combo trees).

41.2 Preliminary Comments on Procedure Map Encapsulation and Expansion

Like other knowledge in CogPrime , procedures may be stored in either a localized (Combo tree) or globalized (procedure map) manner, with the different approaches being appropriate for different purposes. Activation of a localized procedure may spur activation of a globalized procedure, and vice versa – so on the overall mind-network level the representation of procedures is heavily "glocal."

One issue that looms large in this context is the conversion between localized and globalized procedures – i.e., in CogPrime lingo, the encapsulation and expansion of procedure maps. This matter will be considered in more detail in Chapter 42 but here we briefly review some key ideas.

Converting from grounded ProcedureNodes into maps is a relatively simple learning problem: one enacts the procedure, observes which Atoms are active at what times during the enaction process, and then creating PredictiveImplicationLinks between the Atoms active at a certain time and those

active at subsequent times. Generally it will be necessary to enact the procedure multiple times and with different inputs, to build up the appropriate library of PredictiveImplicationLinks.

Converting from maps into ProcedureNodes is significantly trickier. First, it involves carrying out data mining over the network of ProcedureNodes, identifying subnetworks that are coherent schema or predicate maps. Then it involves translating the control structure of the map into explicit logical form, so that the encapsulated version will follow the same order of execution as the map version. This is an important case of the general process of map encapsulation, to be discussed in Chapter 42

Next, the learning of grounded ProcedureNodes is carried out by a synergistic combination of multiple mechanisms, including pure procedure learning methods like hillclimbing and evolutionary learning, and logical inference. These two approaches have quite different characteristics. Evolutionary learning and hillclimbing excel at confronting a problem that the system has no clue about, and arriving at a reasonably good solution in the form of a schema or predicate. Inference excels at deploying the system's existing knowledge to form useful schemata or predicates. The choice of the appropriate mechanism for a given problem instance depends largely on how much relevant knowledge is available.

A relatively simple case of ProcedureNode learning is where one is given a ConceptNode and wants to find a ProcedureNode matching it. For instance, given a ConceptNode C, one may wish to find the simplest possible predicate whose corresponding PredicateNode P satisfies

```
SatisfyingSet(P) = C
```

On the other hand, given a ConceptNode C involved in inferred ExecutionLinks of the form

```
ExecutionLink C Ai Bi
i=1,...,n
```

one may wish to find a SchemaNode so that the corresponding SchemaNode will fulfill this same set of ExecutionLinks. It may seem surprising at first that a ConceptNode might be involved with ExecutionLinks, but remember that a function can be seen as a set of tuples (ListLink in CogPrime) where the first elements, the inputs of the function, are associated with a unique output. These kinds of ProcedureNode learning may be cast as optimization problems, and carried out by hillclimbing or evolutionary programming. Once procedures are learned via evolutionary programming or other techniques, they may be refined via inference.

The other case of ProcedureNode learning is goal-driven learning. Here one seeks a SchemaNode whose execution will cause a given goal (represented by a Goal Node) to be satisfied. The details of Goal Nodes have already been reviewed; but all we need to know here is simply that a Goal Node presents an objective function, a function to be maximized; and that it poses the problem

of finding schemata whose enaction will cause this function to be maximized in specified contexts.

The learning of procedure maps, on the other hand, is carried out by reinforcement learning, augmented by inference. This is a matter of the system learning HebbianLinks between ProcedureNodes, as will be described below.

41.3 Predicate Schematization

Now we turn to the process called "predicate schematization," by which declarative knowledge about how to carry out actions may be translated into Combo trees embodying specific procedures for carrying out actions. This process is straightforward and automatic in some cases, but in other cases requires significant contextually-savvy inference. This is a critical process because some procedure knowledge – especially that which is heavily dependent on context in either its execution or its utility – will be more easily learned via inferential methods than via pure procedure-learning methods. But, even if a procedure is initially learned via inference (or is learned by inference based on cruder initial guesses produced by pure procedure learning methods), it may still be valuable to have this procedure in compact and rapidly executable form such as Combo provides.

To proceed with the technical description of predicate schematization in CogPrime , we first need the notion of an "executable predicate". Some predicates are executable in the sense that they correspond to executable schemata, others are not. There are executable atomic predicates (represented by individual PredicateNodes), and executable predicates (which are link structures). In general, a predicate may be turned into a schema if it is an atomic executable predicate, or if it is a compound link structure that consists entirely of executable atomic predicates (e.g. pick_up, walk_to, can_do, etc.) and temporal links (e.g. SimultaneousAND, PredictiveImplication, etc.)

Records of predicate execution may then be made using ExecutionLinks, e.g.

```
ExecutionLink pick_up ( me, ball_7)
```

is a record of the fact that the schema corresponding to the pick_up predicate was executed on the arguments (me, ball_7).

It is also useful to introduce some special (executable) predicates related to schema execution:

- can_do, which represents the system's perceived ability to do something
- do, which denotes the system actually doing something; this is used to mark actions as opposed to perceptions
- just_done, which is true of a schema if the schema has very recently been executed.

The general procedure used in figuring out what predicates to schematize, in order to create a procedure achieving a certain goal, is: Start from the goal and work backwards, following PredictiveImplications and EventualPredictiveImplications and treating can_do's as transparent, stopping when you find something that can currently be done, or else when the process dwindles due to lack of links or lack of sufficiently certain links.

In this process, an ordered list of perceptions and actions will be created. The Atoms in this perception/action-series (PA-series) are linked together via temporal-logical links.

The subtlety of this process, in general, will occur because there may be many different paths to follow. One has the familiar combinatorial explosion of backward-chaining inference, and it may be hard to find the best PA-series among all the mess. Experience-guided pruning is needed here just as with backward-chaining inference.

Specific rules for translating temporal links into executable schemata, used in this process, are as follows. All these rule-statements assume that B is in the selected PA-series. All node variables not preceded by do or can_do are assumed to be perceptions. The → denotes the transformation from predicates to executable schemata.

```
EventualPredictiveImplicationLink (do A) B

  →

Repeat (do A) Until B


EventualPredictiveImplicationLink (do A) (can_do B)

  →

Repeat
    do A
    do B
Until
    Evaluation just_done B
```

the understanding being that the agent may try to do B and fail, and then try again the next time around the loop

```
PredictiveImplicationLink (do A) (can_do B) <time-lag T>

  →

do A
wait T
do B


SimultaneousImplicationLink A (can_do B)

  →

if A then do B
```

```
SimultaneousImplicationLink (do A) (can_do B)

  →

do A
do B

PredictiveImplicationLink A (can_do B)

  →

if A then do B

SequentialAndLink A1 ... An

  →

A1
...
An

SequentialAndLink A1 ... An <time_lag T>

  →

A1
Wait T
A2
Wait T
...
Wait T
An

SimultaneousANDLink A1 ? An

  →

A1
...
An
```

Note how all instances of can_do are stripped out upon conversion from predicate to schema, and replaced with instances of do.

41.3.1 A Concrete Example

For a specific example of this process, consider the knowledge that: "If I walk to the teacher while whistling, and then give the teacher the ball, I'll get rewarded."

This might be represented by the predicates
walk to the teacher while whistling

```
A_1 :=
SimultaneousAND
    do Walk_to
        ExOutLink locate teacher
    EvaluationLink do whistle
```

If I walk to the teacher while whistling, eventually I will be next to the teacher

```
EventualPredictiveImplication
    A_1
    Evaluation next_to teacher
```

While next to the teacher, I can give the teacher the ball

```
SimultaneousImplication
    EvaluationLink next_to teacher
    can_do
        EvaluationLink give (teacher, ball)
```

If I give the teacher the ball, I will get rewarded

```
PredictiveImplication
    just_done
        EvaluationLink done give (teacher, ball)
    Evaluation reward
```

Via goal-driven predicate schematization, these predicates would become the schemata
walk toward the teacher while whistling

```
Repeat:
    do WalkTo
        ExOut locate teacher
    do Whistle
Until:
    next_to(teacher, ball)
```

if next to the teacher, give the teacher the ball

```
If:
    Evaluation next_to teacher
Then
    do give(teacher, ball)
```

Carrying out these two schemata will lead to the desired behavior of walking toward the teacher while whistling, and then giving the teacher the ball when next to the teacher.

Note that, in this example:

- The walk_to, whistle, locate and give used in the example schemata are procedures corresponding to the executable predicates walk_to, whistle, locate and give used in the example predicates
- Next_to is evaluated rather than executed because (unlike the other atomic predicates in the overall predicate being made executable) it has no "do" or "can_do" next to it

41.4 Concept-Driven Schema and Predicate Creation

In this section we will deal with the "conversion" of ConceptNodes into SchemaNodes or PredicateNodes. The two cases involve similar but nonidentical methods; we will begin with the simpler PredicateNode case. Conceptually, the importance of this should be clear: sometimes knowledge may be gained via concept-learning or linguistic means, but yet may be useful to the mind in other forms, e.g. as executable schema or evaluable predicates. For instance, the system may learn conceptually about bicycle-riding, but then may also want to learn executable procedures allowing it to ride a bicycle. Or it may learn conceptually about criminal individuals, but may then want to learn evaluable predicates allowing it to quickly evaluate whether a given individual is a criminal or not.

41.4.1 Concept-Driven Predicate Creation

Suppose we have a ConceptNode C, with a set of links of the form

```
MemberLink A_i C, i=1,...,n
```

Our goal is to find a PredicateNode so that firstly,

```
MemberLink X C

is equivalent to

X ''within'' SatisfyingSet(P)
```

and secondly,

```
P is as simple as possible
```

This is related to the "Occam's Razor," Solomonoff induction related heuristic to be presented later in this chapter.

We now have an optimization problem: search the space of predicates for P that maximize the objective function f(P,C), defined as for instance

$$f(P, C) = cp(P) \times r(C, P)$$

where $cp(P)$, the complexity penalty of P, is a positive function that decreases when P gets larger and with $r(C, P) =$

```
GetStrength
    SimilarityLink
        C
        SatisfyingSet(P)
```

This is an optimization problem over predicate space, which can be solved in an approximate way by the evolutionary programming methods described earlier.

The ConceptPredicatization MindAgent selects ConceptNodes based on

- Importance
- Total (truth value based) weight of attached MemberLinks and EvaluationLinks

and launches an evolutionary learning or hillclimbing task focused on learning predicates based on the nodes it selects.

41.4.2 Concept-Driven Schema Creation

In the schema learning case, instead of a ConceptNode with MemberLinks and EvaluationLinks, we begin with a ConceptNode C with ExecutionLinks. These ExecutionLinks were presumably produced by inference (the only CogPrime cognitive process that knows how to create ExecutionLinks for non-ProcedureNodes).

The optimization problem we have here is: search the space of schemata for S that maximize the objective function $f(S, C)$, defined as follows:

$$f(S, C) = cp(S) \times r(S, C)$$

Let $Q(C)$ be the set of pairs (X, Y) so that $ExecutionLink\ C\ X\ Y$, and

```
r(S,C) =

GetStrength
    SubsetLink
        Q(C)
        Graph(S)
```

where $Graph(S)$ denotes the set of pairs (X, Y) so that $ExecutionLink\ S\ X\ Y$, where S has been executed over all valid inputs.

Note that we consider a SubsetLink here because in practice C would have been observed on a partial set of inputs.

Operationally, the situation here is very similar to that with concept predicatization. The ConceptSchematization MindAgent must select ConceptNodes based on:

- Importance
- Total (truth value based) weight of ExecutionLinks

and then feed these to evolutionary optimization or hillclimbing.

41.5 Inference-Guided Evolution of Pattern-Embodying Predicates

Now we turn to predicate learning – the learning of PredicateNodes, in particular.

Aside from logical inference and learning predicates to match existing concepts, how does the system create new predicates? Goal-driven schema learning (via evolution or reinforcement learning) provides one alternate approach: create predicates in the context of creating useful schema. Pattern mining, discussed in Chapter 37, provides another. Here we will describe (yet) another complementary dynamic for predicate creation: pattern-oriented, inference-guided PredicateNode evolution.

In most general terms, the notion pursued here is to form predicates that embody patterns in itself and in the world. This brings us straight back to the foundations of the patternist philosophy of mind, in which mind is viewed as a system for recognizing patterns in itself and in the world, and then embodying these patterns in itself. This general concept is manifested in many ways in the CogPrime design, and in this section we will discuss two of them:

- Reward of surprisingly probable Predicates
- Evolutionary learning of pattern-embodying Predicates

These are emphatically not the only way pattern-embodying PredicateNodes get into the system. Inference and concept-based predicate learning also create PredicateNodes embodying patterns. But these two mechanisms complete the picture.

41.5.1 Rewarding Surprising Predicates

The TruthValue of a PredicateNode represents the expected TruthValue obtained by averaging its TruthValue over all its possible legal argument-values. Some Predicates, however, may have high TruthValue without really being *worthwhile*. They may not add any information to their components. We want to identify and reward those Predicates whose TruthValues actually add information beyond what is implicit in the simple fact of combining their components.

For instance, consider the PredicateNode

```
AND
   InheritanceLink X man
   InheritanceLink X ugly
```

If we assume the man and ugly concepts are independent, then this PredicateNode will have the TruthValue

$$man.tv.s \times ugly.tv.s$$

In general, a PredicateNode will be considered interesting if:

1. Its Links are important
2. Its TruthValue differs significantly from what would be expected based on independence assumptions about its components

It is of value to have interesting Predicates allocated more attention than uninteresting ones. Factor 1 is already taken into account, in a sense: if the PredicateNode is involved in many Links this will boost its activation which will boost its importance. On the other hand, Factor 2 is not taken into account by any previously discussed mechanisms.

For instance, we may wish to reward a PredicateNode if it has a surprisingly large or small strength value. One way to do this is to calculate:

$$sdiff = |actual\ strength - strength\ predicted\ via\ independence\ assumptions| \times weight\_of\_evidence$$

and then increment the value:

$$K \times sdiff$$

onto the PredicateNode's LongTermImportance value, and similarly increment STI using a different constant.

Another factor that might usefully be caused to increment LTI is the simplicity of a PredicateNode. Given two Predicates with equal strength, we want the system to prefer the simpler one over the more complex one. However, the OccamsRazor MindAgent, to be presented below, rewards simpler Predicates directly in their strength values. Hence if the latter is in use, it seems unnecessary to reward them for their simplicity in their LTI values as well. This is an issue that may require some experimentation as the system develops.

Returning to the surprisingness factor, consider the PredicateNode representing

```
AND
    InheritanceLink X cat
    EvaluationLink (eats X) fish
```

If this has a surprisingly high truth value, this means that there are more X known to (or inferred by) the system, that both inherit from *cat* and eat fish, than one would expect given the probabilities of a random X both inheriting from cat and eating fish. Thus, roughly speaking, the conjunction of inheriting from cat and eating fish may be a pattern in the world.

We now see one very clear sense in which CogPrime dynamics implicitly leads to predicates representing patterns. Small predicates that have surprising truth values get extra activation, hence are more likely to stick around in the system. Thus the mind fills up with patterns.

41.5.2 A More Formal Treatment

It is worth taking a little time to clarify the sense in which we have a *pattern* in the above example, using the mathematical notion of pattern reviewed in Chapter 3 of Part 1.

Consider the predicate:

```
pred1(T).tv
equals
        >
           GetStrength
               AND
                    Inheritance $X cat
                    Evaluation eats ($X, fish)
           T
```

where T is some threshold value (e.g. .8). Let B = SatisfyingSet(pred1(T)). B is the set of everything that inherits from cat and eats fish.

Now we will make use of the notion of *basic complexity*. If one assumes the entire AtomSpace A constituting a given CogPrime system as given background information, then the basic complexity $c(B||A)$ may be considered as the number of bits required to list the handles of the elements of B, for lookup in A; whereas $c(B)$ is the number of bits required to actually list the elements of B. Now, the formula given above, defining the set B, may be considered as a process P whose output is the set B. The simplicity $c(P||A)$ is the number of bits needed to describe this process, which is a fairly small number. We assume A is given as background information, accessible to the process.

Then the degree to which P is a pattern in B is given by

$$1 - c(P||A)/c(B||A)$$

which, if B is a sizable category, is going to be pretty close to 1.

The key to there being a pattern here is that the relation:

```
(Inheritance X cat) AND (eats X fish)
```

has a high strength and also a high count. The high count means that B is a large set, either by direct observation or by hypothesis (inference). In the case where the count represents actual pieces of evidence observed by the system and retained in memory, then quite literally and directly, the PredicateNode represents a pattern in a subset of the system (relative to the

background knowledge consisting of the system as a whole). On the other hand, if the count value has been obtained indirectly by inference, then it is possible that the system does not actually know any examples of the relation. In this case, the PredicateNode is not a pattern in the actual memory store of the system, but it is being hypothesized to be a pattern in the world in which the system is embedded.

41.6 PredicateNode Mining

We have seen how the natural dynamics of the CogPrime system, with a little help from special heuristics, can lead to the evolution of Predicates that embody patterns in the system's perceived or inferred world. But it is also valuable to more aggressively and directly create pattern-embodying Predicates. This does not contradict the implicit process, but rather complements it. The explicit process we use is called *PredicateNode Mining* and is carried out by a PredicateNodeMiner MindAgent.

Define an Atom structure template as a schema expression corresponding to a CogPrime Link in which some of the arguments are replaced with variables. For instance,

```
Inheritance X cat
```

```
EvaluationLink (eats X) fish
```

are Atom structure templates. (Recall that Atom structure templates are important in PLN inference control, as reviewed in 36)

What the PredicateNodeMiner does is to look for Atom structure templates and logical combinations thereof which

- Minimize PredicateNode size
- Maximize surprisingness of truth value

This is accomplished by a combination of heuristics.

The first step in PredicateNode mining is to find Atom structure templates with high truth values. This can be done by a fairly simple heuristic search process.

First, note that if one specifies an (Atom, Link type), one is specifying a set of Atom structure templates. For instance, if one specifies

```
(cat, InheritanceLink)
```

then one is specifying the templates

```
InheritanceLink $X cat
```

and

```
InheritanceLink cat $X
```

One can thus find Atom structure templates as follows. Choose an Atom with high truth value, and then, for each Link type, tabulate the total truth value of the Links of this type involving this Atom. When one finds a promising (Atom, Link type) pair, one can then do inference to test the truth value of the Atom structure template one has found.

Next, given high-truth-value Atom structure templates, the PredicateNodeMiner experiments with joining them together using logical connectives. For each potential combination it assesses the fitness in terms of size and surprisingness. This may be carried out in two ways:

1. By incrementally building up larger combinations from smaller ones, at each incremental stage keeping only those combinations found to be valuable
2. For large combinations, by evolution of combinations

Option 1 is basically greedy data mining (which may be carried out via various standard algorithms, as discussed in Chapter 37), which has the advantage of being much more rapid than evolutionary programming, but the disadvantage that it misses large combinations whose subsets are not as surprising as the combinations themselves. It seems there is room for both approaches in CogPrime (and potentially many other approaches as well). The PredicateNodeMiner MindAgent contains a parameter telling it how much time to spend on stochastic pattern mining vs. evolution, as well as parameters guiding the processes it invokes.

So far we have discussed the process of finding single-variable Atom structure templates. But multivariable Atom structure templates may be obtained by combining single-variable ones. For instance, given

```
eats $X fish
```

```
lives_in $X Antarctica
```

one may choose to investigate various combinations such as

```
(eats $X $Y) AND (lives_in $X $Y)
```

(this particular example will have a predictably low truth value). So, the introduction of multiple variables may be done in the same process as the creation of single-variable combinations of Atom structure templates.

When a suitably fit Atom structure template or logical combination thereof is found, then a PredicateNode is created embodying it, and placed into the AtomSpace. WIKISOURCE:SchemaMaps

41.7 Learning Schema Maps

Next we plunge into the issue of procedure maps - schema maps in particular. A schema map is a simple yet subtle thing - a subnetwork of the AtomSpace

consisting of SchemaNodes, computing some useful quantity or carrying out some useful process in a cooperative way. The general purpose of schema maps is to allow schema execution to interact with other mental processes in a more flexible way than is allowed by compact Combo trees with internal hooks into the AtomSpace. I.e., to handle cases where procedure execution needs to be *very* highly interactive, mediated by attention allocation and other CogPrime dynamics in a flexible way.

But how can schema maps be learned? The basic idea is simply reinforcement learning. In a goal-directed system consisting of interconnected, cooperative elements, you reinforce those connections and/or those elements that have been helpful for achieving goals, and weaken those connections that haven't. Thus, over time, you obtain a network of elements that achieves goals effectively.

The central difficulty in all reinforcement learning approaches is the 'assignment of credit' problem. If a component of a system has been directly useful for achieving a goal, then rewarding it is easy. But if the relevance of a component to a goal is indirect, then things aren't so simple. Measuring indirect usefulness in a large, richly connected system is difficult - inaccuracies creep into the process easily.

In CogPrime , reinforcement learning is handled via HebbianLinks, acted on by a combination of cognitive processes. Earlier, in Chapter 23, we reviewed the semantics of HebbianLinks, and discussed two methods for forming HebbianLinks:

1. Updating HebbianLink strengths via mining of the System Activity Table
2. Logical inference on HebbianLinks, which may also incorporate the use of inference to combine HebbianLinks with other logical links (for instance, in the reinforcement learning context, PredictiveImplicationLinks)

We now describe how HebbianLinks, formed and manipulated in this manner, may play a key role in goal-driven reinforcement learning. In effect, what we will describe is an implicit integration of the bucket brigade with PLN inference. The addition of robust probabilistic inference adds a new kind of depth and precision to the reinforcement learning process.

Goal Nodes have an important ability to stimulate a lot of SchemaNode execution activity. If a goal needs to be fulfilled, it stimulates schemata that are known to make this happen. But how is it known which schemata tend to fulfill a given goal? A link:

```
PredictiveImplicationLink S G
```

means that after schema S has been executed, goal G tends to be fulfilled. If these links between goals and goal-valuable schemata exist, then activation spreading from goals can serve the purpose of causing goal-useful schemata to become active.

The trick, then, is to use HebbianLinks and inference thereon to implicitly guess PredictiveImplicationLinks. A HebbianLink between S1 and S says that

when thinking about S1 was useful in the past, thinking about S was also often useful. This suggests that if doing S achieves goal G, maybe doing S1 is also a good idea. The system may then try to find (by direct lookup or reasoning) whether, in the current context, there is a PredictiveImplication joining S1 to S. In this way Hebbian reinforcement learning is being used as an inference control mechanism to aid in the construction of a goal-directed chain of PredictiveImplicationLinks, which may then be schematized into a contextually useful procedure.

Note finally that this process feeds back into itself in an interesting way, via contributing to ongoing HebbianLink formation. Along the way, while leading to the on-the-fly construction of context-appropriate procedures that achieve goals, it also reinforces the HebbianLinks that hold together schema maps, sculpting new schema maps out of the existing field of interlinked SchemaNodes.

41.7.1 Goal-Directed Schema Evolution

Finally, as a complement to goal-driven reinforcement learning, there is also a process of goal-directed SchemaNode learning. This combines features of the goal-driven reinforcement learning and concept-driven schema evolution methods discussed above. Here we use a Goal Node to provide the fitness function for schema evolution.

The basic idea is that the fitness of a schema is defined by the degree to which enactment of that schema causes fulfillment of the goal. This requires the introduction of CausalImplicationLinks, as defined in PLN. In the simplest case, a CausalImplicationLink is simply a PredictiveImplicationLink.

One relatively simple implementation of the idea is as follows. Suppose we have a Goal Node G, whose satisfaction we desire to have achieved by time T1. Suppose we want to find a SchemaNode S whose execution at time T2 will cause G to be achieved. We may define a fitness function for evaluating candidate S by:

$$f(S, G, T1, T2) = cp(S) \times r(S, G, T1, T2)$$

```
r(S,G,T1,T2) =
    GetStrength
        CausalImplicationLink
            EvaluationLink
                AtTime
                    T1
                    ExecutionLink S X Y
            EvaluationLink AtTime (T2, G)
```

Another variant specifies only a relative time lag, not two absolute times.

$$f(S,G,T) = cp(S) \times v(S,G,T)$$

```
v(S,G,T) =
     AND
          NonEmpty
              SatisfyingSet r(S,G,T1,T2)
          T1 > T2 - T
```

Using evolutionary learning or hillclimbing to find schemata fulfilling these fitness functions, results in SchemaNodes whose execution is expected to cause the achievement of given goals. This is a complementary approach to reinforcement-learning based schema learning, and to schema learning based on PredicateNode concept creation. The strengths and weaknesses of these different approaches need to be extensively experimentally explored. However, prior experience with the learning algorithms involved gives us some guidance.

We know that when absolutely nothing is known about an objective function, evolutionary programming is often the best way to proceed. Even when there is knowledge about an objective function, the evolution process can take it into account, because the fitness functions involve logical links, and the evaluation of these logical links may involve inference operations.

On the other hand, when there's a lot of relevant knowledge embodied in previously executed procedures, using logical reasoning to guide new procedure creation can be cumbersome, due to the overwhelming potentially useful number of facts to choose when carrying inference. The Hebbian mechanisms used in reinforcement learning may be understood as inferential in their conceptual foundations (since a HebbianLink is equivalent to an ImplicationLink between two propositions about importance levels). But in practice they provide a much-streamlined approach to bringing knowledge implicit in existing procedures to bear on the creation of new procedures. Reinforcement learning, we believe, will excel at combining existing procedures to form new ones, and modifying existing procedures to work well in new contexts. Logical inference can also help here, acting in cooperation with reinforcement learning. But when the system has no clue how a certain goal might be fulfilled, evolutionary schema learning provides a relatively time-efficient way for it to find something minimally workable.

Pragmatically, the GoalDrivenSchemaLearning MindAgent handles this aspect of the system's operations. It selects Goal Nodes with probability proportional to importance, and then spawns problems for the Evolutionary Optimization Unit Group accordingly. For a given Goal Node, PLN control mechanisms are used to study its properties and select between the above objective functions to use, on an heuristic basis.

41.8 Occam's Razor

Finally we turn to an important cognitive process that fits only loosely into the category of "CogPrime Procedure learning" — it's not actually a *procedure learning* process, but rather a process that utilizes the fruits of procedure learning.

The well-known "Occam's razor" heuristic says that all else being equal, simpler is better. This notion is embodied mathematically in the Solomonoff-Levin "universal prior," according to which the a priori probability of a computational entity X is defined as a normalized version of:

$$m(X) = \sum_p 2^{-l(p)}$$

where:

- the sum is taken over all programs p that compute X
- l(p) denotes the length of the program p

Normalization is necessary because these values will not automatically sum to 1 over the space of all X.

Without normalization, m is a semimeasure rather than a measure; with normalization it becomes the "Solomonoff-Levin measure" [Lev94].

Roughly speaking, Solomonoff's induction theorem [**?**, Sol64a] shows that, if one is trying to learn the computer program underlying a given set of observed data, and one does Bayesian inference over the set of all programs to try and obtain the answer, then if one uses the universal prior distribution one will arrive at the correct answer.

CogPrime is not a Solomonoff induction engine. The computational cost of actually applying Solomonoff induction is unrealistically large. However, as we have seen in this chapter, there are aspects of CogPrime that are reminiscent of Solomonoff induction. In concept-directed schema and predicate learning, in pattern-based predicate learning - and in causal schema learning, we are searching for schemata and predicates that minimize complexity while maximizing some other quality. These processes all implement the Occam's Razor heuristic in a Solomonoffian style.

Now we will introduce one more method of imposing the heuristic of algorithmic simplicity on CogPrime Atoms (and hence, indirectly, on CogPrime maps as well). This is simply to give a higher a priori probability to entities that are more simply computable.

For starters, we may increase the node probability of ProcedureNodes proportionately to their simplicity. A reasonable formula here is simply:

$$2^{-rc(P)}$$

where P is the ProcedureNode and $r > 0$ is a parameter. This means that infinitely complex P have a priori probability zero, whereas an infinitely simple P has an a priori probability 1.

This is not an exact implementation of the Solomonoff-Levin measure, but it's a decent heuristic approximation. It is not pragmatically realistic to sum over the lengths of all programs that do the same thing as a given predicate P. Generally the first term of the Solomonoff-Levin summation is going to dominate the sum anyway, so if the ProcedureNode P is maximally compact, then our simplified formula will be a good approximation of the Solomonoff-Levin summation. These apriori probabilities may be merged with node probability estimates from other sources, using the revision rule.

A similar strategy may be taken with ConceptNodes. We want to reward a ConceptNode C with a higher apriori probability if $C \in \mathit{SatisfyingSet}(P)$ for a simple PredicateNode P. To achieve this formulaically, let $sim(X, Y)$ denote the strength of the SimilarityLink between X and Y, and let:

$$sim'(C, P) = sim(C, \mathit{SatisfyingSet}(P))$$

We may then define the apriori probability of a ConceptNode as:

$$pr(C) = \sum_P sim'(C, P) 2^{-rc(P)}$$

where the sum goes over all P in the system. In practice of course it's only necessary to compute the terms of the sum corresponding to P so that $sim'(C, P)$ is large.

As with the a priori PredicateNode probabilities discussed above, these apriori ConceptNode probabilities may be merged with other node probability information, using the revision rule, and using a default parameter value for the weight of evidence. There is one pragmatic difference here from the PredicateNode case, though. As the system learns new PredicateNodes, its best estimate of $pr(C)$ may change. Thus it makes sense for the system to store the apriori probabilities of ConceptNodes separately from the node probabilities, so that when the apriori probability is changed, a two step operation can be carried out:

- First, remove the old apriori probability from the node probability estimate, using the reverse of the revision rule
- Then, add in the new apriori probability

Finally, we can take a similar approach to any Atom Y produced by a SchemaNode. We can construct:

$$pr(Y) = \sum_{S,X} s(S, X, Y) 2^{-r(c(S)+c(X))}$$

where the sum goes over all pairs (S, X) so that:

```
ExecutionLink S X Y
```

and $s(S, X, Y)$ is the strength of this ExecutionLink. Here, we are rewarding Atoms that are produced by simple schemata based on simple inputs.

The combined result of these heuristics is to cause the system to prefer simpler explanations, analysis, procedures and ideas. But of course this is only an apriori preference, and if more complex entities prove more useful, these will quickly gain greater strength and importance in the system.

Implementationally, these various processes are carried out by the OccamsRazor MindAgent. This dynamic selects ConceptNodes based on a combination of:

- importance
- time since the apriori probability was last updated (a long time is preferred)

It selects ExecutionLinks based on importance and based on the amount of time since they were last visited by the OccamsRazor MindAgent. And it selects PredicateNodes based on importance, filtering out PredicateNodes it has visited before.

Chapter 42
Map Formation

Abstract

42.1 Introduction

In Chapter [**?**] we distinguished the explicit versus implicit aspects of knowledge representation in CogPrime . The explicit level consists of Atoms with clearly comprehensible meanings, whereas the implicit level consists of "maps" – collections of Atoms that become important in a coordinated manner, analogously to cell assemblies in an attractor neural net. The combination of the two is valuable because the world-patterns useful to human-like minds in achieving their goals, involve varying degrees of isolation and interpenetration, and their effective goal-oriented processing involves both symbolic manipulation (for which explicit representation is most valuable) and associative creative manipulation (for which distributed, implicit representation is most valuable).

The chapters since have focused primarily on explicit representation, commenting on the implicit "map" level only occasionally. There are two reasons for this: one theoretical, one pragmatic. The theoretical reason is that the majority of map dynamics and representations are implicit in Atom-level correlates. And the pragmatic reason is that, at this stage, we simply do not know as much about CogPrime maps as we do about CogPrime Atoms. Maps are emergent entities and, lacking a detailed theory of CogPrime dynamics, the only way we have to study them in detail is to run CogPrime systems and mine their System Activity Tables and logs for information. If CogPrime research goes well, then updated versions of this book may include more details on observed map dynamics in various contexts.

In this chapter, however, we finally turn our gaze directly to maps and their relationships to Atoms, and discuss processes that convert Atoms into maps (expansion) and vice versa (encapsulation). These processes represent

a bridge between the concretely-implemented and emergent aspects of CogPrime 's mind.

Map encapsulation is the process of recognizing Atoms that tend to become important in a coordinated manner, and then creating new Atoms grouping these. As such it is essentially a form of AtomSpace pattern mining. In terms of patternist philosophy, map encapsulation is a direct incarnation of the so-called "cognitive equation"; that is, the process by which the mind recognizes patterns in itself, and then embodies these patterns as new content within itself – an instance of what Hofstadter famously labeled a "strange loop" [Hof79]. In SMEPH terms, the encapsulation process is how CogPrime explicitly studies its own derived hypergraph and then works to implement this derived hypergraph more efficiently by recapitulating it at the concretely-implemented-mind level. This of course may change the derived hypergraph considerably. Among other things, map encapsulation has the possibility of taking the things that were the most abstract, *highest level* patterns in the system and forming new patterns involving them and their interrelationships – thus building the highest level of patterns in the system higher and higher. Figures 42.2 and 42.1 illustrate concrete examples of the process.

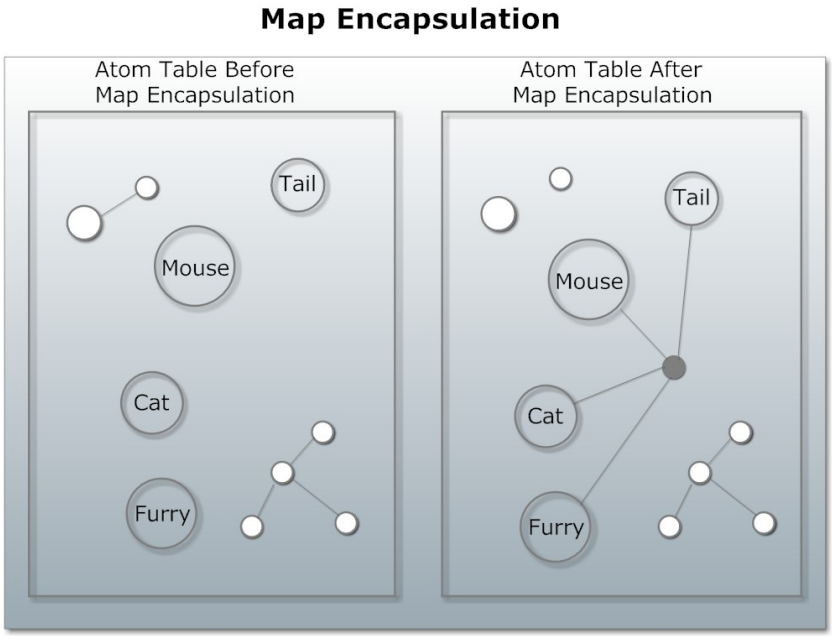


Fig. 42.1 Illustration of the process of creating explicit Atoms corresponding to a pattern previously represented as a distributed "map."

Map expansion, on the other hand, is the process of taking knowledge that is explicitly represented and causing the AtomSpace to represent it *implicitly*,

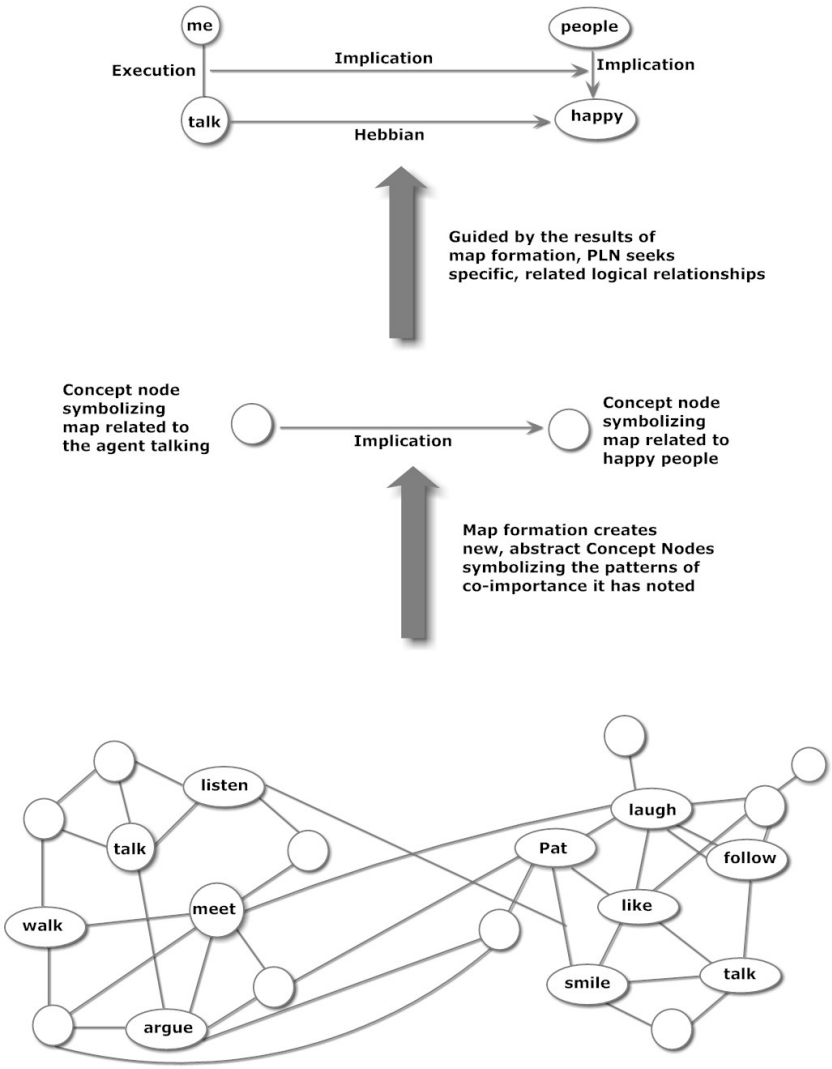


Fig. 42.2 Illustration of the process of creating explicit Atoms corresponding to a pattern previously represented as a distributed "map."

on the map level. In many cases this will happen automatically. For instance, a ConceptNode C may turn into a concept map if the importance updating process iteratively acts in such a way as to create/reinforce a map consisting of C and its relata. Or, an Atom-level InheritanceLink may implicitly spawn a map-level InheritanceEdge (in SMEPH terms). However, there is one important case in which Atom-to-map conversion must occur explicitly: the expansion of compound ProcedureNodes into procedure maps. This must

occur explicitly because the process graphs inside ProcedureNodes have no dynamics going on except evaluation; there is no opportunity for them to manifest themselves as maps, unless a MindAgent is introduced that explicitly does so. Of course, just unfolding a Combo tree into a procedure map doesn't intrinsically make it a significant part of the derived hypergraph - but it opens the door for the inter-cognitive-process integration that may make this occur.

42.2 Map Encapsulation

Returning to encapsulation: it may be viewed as a form of *symbolization,* in which the system creates concrete entities to serve as symbols for its own emergent patterns. It can then study an emergent pattern's interrelationships by studying the interrelationships of the symbol with other symbols.

For instance, suppose a system has three derived-hypergraph ConceptVertices A, B and C, and observes that:

```
InheritanceEdge A B
InheritanceEdge B C
```

Then encapsulation may create ConceptNodes A', B' and C' for A, B and C, and InheritanceLinks corresponding to the InheritanceEdges, where e.g. A' is a set containing all the Atoms contained in the static map A. First-order PLN inference will then immediately conclude:

```
InheritanceLink A' C'
```

and it may possibly do so with a higher strength than the strength corresponding to the (perhaps not significant) InheritanceEdge between A and C. But if the encapsulation is done right then the existence of the new InheritanceLink will indirectly cause the formation of the corresponding:

```
InheritanceEdge A C
```

via the further action of inference, which will use (InheritanceLink A' C') to trigger the inference of further inheritance relationships between members of A' and members of C', which will create an emergent inheritance between members of A (the map corresponding to A') and C (the map corresponding to C').

The above example involved the conversion of static maps into ConceptNodes. Another approach to map encapsulation is to represent the fact that a set of Atoms constitutes a map as a predicate; for instance if the nodes A, B and C are habitually used together, then the predicate P may be formed, where:

```
P =
AND
   A is used at time T
```

```
B is used at time T
C is used at time T
```

The habitualness of A, B and C being used together will be reflected in the fact that P has a surprisingly high truth value. By a simple concept formation heuristic, this may be used to form a link AND(A, B, C), so that:

```
AND(A, B, C) is used at time T
```

This composite link AND(A, B, C) is then an embodiment of the map in single-Atom form.

Similarly, if a set of schemata is commonly used in a certain series, this may be recognized in a predicate, and a composite schema may then be created embodying the component schemata. For instance, suppose it is recognized as a pattern that:

```
AND
   S1 is used at time T on input I1 producing output O1
   S2 is used at time T+s on input O1 producing output O2
```

Then we may explicitly create a schema that consists of S1 taking input and feeding its output to S2. This cannot be done via any standard concept formation heuristic; it requires a special process.

One might wonder why this Atom-to-map conversion process is necessary: Why not just let maps combine to build new maps, hierarchically, rather than artificially transforming some maps into Atoms and letting maps then form from these map-representing Atoms. It is all a matter of precision. Operations on the map level are fuzzier and less reliable than operations on the Atom level. This fuzziness has its positive and its negative aspects. For example, it is good for spontaneous creativity, but bad for constructing lengthy, confident chains of thought. WIKISOURCE:ActivityTables

42.3 Atom and Predicate Activity Tables

A major role in map formation is played by a collection of special tables. Map encapsulation takes place, not by data mining directly on the AtomTable, but by data mining on these special tables constructed from the AtomTable, specifically with efficiency of map mining in mind.

First, there is the Atom Utilization Table, which may be derived from the SystemActivityTable. The Atom Utilization Table, in its most simple possible version, takes the form

The calculation of "utility" values for this purpose must be done in a "local" way by MindAgents, rather than by a global calculation of the degree to which utilizing a certain Atom has led to the achievement of a certain system goal (this kind of global calculation would be better in principle, but it would require massive computational effort to calculate for every Atom in the system at frequent intervals). Each MindAgent needs to estimate how much utility

Time		Atom Handle H
?	?	?
T	?	(Effort spent on Atom H at time t, utility derived from atom H at time t)
?	?	?

Table 42.1 Atom Utilization Table

it has obtained from a given Atom, as well as how much effort it has spent on this Atom, and report these numbers to the Atom Utilization Table.

The normalization of effort values is simple, since effort can be quantified in terms of time and space expended. Normalization of utility values is harder, as it is difficult to define a common scale to span all the different MindAgents, which in some cases carry out very different sorts of operations. One reasonably "objective" approach is to assign each MindAgent an amount of "utility credit", at time T, equal to the amount of currency that the MindAgent has spent since it last disbursed its utility credits. It may then divide up its utility credit among the Atoms it has utilized. Other reasonable approaches may also be defined.

The use of utility and utility credit for Atoms and MindAgents is similar to the stimulus used in the Attention allocation system. There, MindAgents reward Atoms with stimulus to indicate that their short and long term importance should be increased. Merging utility and stimulus is a natural approach to implementing utility in OpenCogPrime .

Note that there are many practical manifestations that the abstract notion of an ActivityTable may take. It could be an ordinary row-and-column style table, but that is not the only nor the most interesting possibility. An ActivityTable may also be effectively stored as a series of graphs corresponding to time intervals - one graph for each interval, consisting of HebbianLinks formed solely based on importance during that interval. In this case it is basically a set of graphs, which may be stored for instance in an AtomTable, perhaps with a special index.

Then there is the Procedure Activity Table, which records the inputs and outputs associated with procedures:

Time		ProcedureNode Handle H
?	?	?
T	?	(Inputs to H, Outputs from H)
?	?	?

Table 42.2 Procedure Activity Table for a Particular MindAgent

Data mining on these tables may be carried out by a variety of algorithms (see MapMining) - the more advanced the algorithm, the fuller the transfer from the derived-hypergraph level to the concretely-implemented level. There is a tradeoff here similar to that with attention allocation - if too much time is

spent studying the derived hypergraph, then there will not be any interesting cognitive dynamics going on anymore because other cognitive processes get no resources, so the map encapsulation process will fail because there is nothing to study!

These same tables may be used in the attention allocation process, for assigning of MindAgent-specific AttentionValues to Atoms. WIKISOURCE:MapMining

42.4 Mining the AtomSpace for Maps

Searching for general maps in a complex AtomSpace is an unrealistically difficult problem, as the search space is huge. So, the bulk of map-mining activity involves looking for the most simple and obvious sorts of maps. A certain amount of resources may also be allocated to looking for subtler maps using more resource-intensive methods.

The following categories of maps can be searched for at relatively low cost:

- Static maps
- Temporal motif maps

Conceptually, a static map is simply a set of Atoms that all tend to be active at the same time.

Next, by a "temporal motif map" we mean a set of pairs:

$$(A_i, t_i)$$

of the type:

$$(Atom, int)$$

so that for many activation cycle indices T, A_i is highly active at some time very close to index $T + t_i$. The reason both static maps and temporal motif maps are easy to recognize is that they are both simply repeated patterns.

Perceptual context formation involves a special case of static and temporal motif mining. In perceptual context formation, one specifically wishes to mine maps involving perceptual nodes associated with a single interaction channel (see Chapter 26 for interaction channel). These maps then represent real-world contexts, that may be useful in guiding real-world-oriented goal activity (via schema-context-goal triads).

In CogPrime so far we have considered three broad approaches for mining static and temporal motif maps from AtomSpaces:

- Frequent subgraph mining, frequent itemset mining, or other sorts of datamining on Activity Tables
- Clustering on the network of HebbianLinks
- Evolutionary Optimization based datamining on Activity Tables

The first two approaches are significantly more time-efficient than the latter, but also significantly more limited in the scope of patterns they can find.

Any of these approaches can be used to look for maps subject to several types of constraints, such as for instance:

- **Unconstrained**: maps may contain any kinds of Atoms
- **Strictly constrained**: maps may only contain Atom types contained on a certain list
- **Probabilistically constrained**: maps must contain Atom types contained on a certain list, as x% of their elements
- **Trigger-constrained**: the map must contain an Atom whose type is on a certain list, as its most active element

Different sorts of constraints will lead to different sorts of maps, of course. We don't know at this stage which sorts of constraints will yield the best results. Some special cases, however, are reasonably well understood. For instance:

- procedure encapsulation, to be discussed below, involves searching for (strictly-constrained) maps consisting solely of ProcedureInstanceNodes.
- to enhance goal achievement, it is likely useful to search for trigger-constrained maps triggered by Goal Nodes.

What the MapEncapsulation CIM-Dynamic (Concretely-Implemented-Mind-Dynamic, see Chapter 19) does once it finds a map, is dependent upon the type of map it's found. In the special case of procedure encapsulation, it creates a compound ProcedureNode (selecting SchemaNode or PredicateNode based on whether the output is a TruthValue or not). For static maps, it creates a ConceptNode, which links to all members of the map with MemberLinks, the weight of which is determined by the degree of map membership. For dynamic maps, it creates PredictiveImplication links depicting the pattern of change.

42.4.1 Frequent Itemset Mining for Map Mining

One class of technique that is useful here is *frequent itemset mining* (FIM), a process that looks to find all frequent combinations of items occurring in a set of data. Another useful class of algorithms is greedy or stochastic itemset mining, which does roughly the same thing as FIM but without being completely exhaustive (the advantage being greater execution speed). Here we will discuss FIM, but the basic concepts are the same if one is doing greedy or stochastic mining instead.

The basic goal of frequent itemset mining is to discover frequent subsets in a group of items. One knows that for a set of N items, there are 2N-1 possible subgroups. To avoid the exponential explosion of subsets, one may compute

the frequent *itemsets* in several rounds. Round i computes all frequent i-itemsets.

A round has two steps: candidate generation and candidate counting. In the candidate generation step, the algorithm generates a set of candidate i-itemsets whose support - a minimum percentage of events in which the item must appear - has not been yet been computed. In the candidate-counting step, the algorithm scans its memory database, counting the support of the candidate itemsets. After the scan, the algorithm discards candidates with support lower than the specified minimum (an algorithm parameter) and retains only the frequent i-itemsets. The algorithm reduces the number of tested subsets by pruning apriori those candidate itemsets that cannot be frequent, based on the knowledge about infrequent itemsets obtained from previous rounds. So for instance if $\{A, B\}$ is a frequent 2-itemset then $\{A, B, C\}$ may possibly be a 3-itemset, on the contrary if $\{A, B\}$ is not a frequent itemset then $\{A, B, C\}$, as well as any super set of $\{A, B\}$, will be discarded. Although the worst case of this sort of algorithm is exponential, practical executions are generally fast, depending essentially on the support limit.

To apply this kind of approach to search for static maps, one simply creates a large set of sets of Atoms – one set for each time-point. In the set S(t) corresponding to time t, we place all Atoms that were firing activation at time t. The itemset miner then searches for sets of Atoms that are subsets of many different S(t) corresponding to many different times t. These are Atom sets that are frequently co-active.

Table **??** presents a typical example of data prepared for frequent itemset mining, in the context of context formation via static-map recognition. Columns represent important nodes and rows indicate time slices. For simplicity, we have *thresholded* the values and show only activity values; so that a 1 in a cell indicates that the Atom indicated by the column was being utilized at the time indicated by the row.

In the example, if we assume minimum support as 50 percent, the context nodes C1 = {Q, R}, and C2 = {Q, T, U} would be created.

Using frequent itemset mining to find temporal motif maps is a similar, but slightly more complex process. Here, one fixes a time-window W. Then, for each activation cycle index t, one creates a set S(t) consisting of pairs of the form:

```
(A, s)
```

where A is an Atom and $0 \leq s \leq W$ is an integer temporal offset. We have:

```
(A,s) ''within'' S(t)
```

if Atom A is firing activation at time t+s. Itemset mining is then used to search for common subsets among the S(t). These common subsets are common patterns of temporal activation, i.e. repeated temporal motifs.

The strength of this approach is its ability to rapidly search through a huge space of possibly significant subsets. Its weakness is its restriction to finding maps that can be incrementally built up from smaller maps. How significant

this weakness is, depends on the particular statistics of map occurrence in CogPrime . Intuitively, we believe frequent itemset mining can perform rather well in this context, and our preliminary experiments have supported this intuition.

Frequent Subgraph Mining for Map Mining

A limitation of FIM techniques, from a CogPrime perspective, is that they are intended for relational databases; but the information about co-activity in a CogPrime instance is generally going to be more efficiently stored as graphs rather than RDB's. Indeed an ActivityTable may be effectively stored as a series of graphs corresponding to time intervals - one graph for each interval, consisting of HebbianLinks formed solely based on importance during that interval. From an ActivityTable stores like this, the way to find maps is not frequent itemset mining but rather frequent subgraph mining, a variant of FIM that is conceptually similar but algorithmically more subtle, and on which there has arisen a significant literature in recent years. We have already briefly discussed this technology in Chapter 37 on pattern mining the Atomspace – map mining being an important special case of Atomspace pattern mining. As noted there, some of the many approaches to frequent subgraph mining are described in [HWP03, KK01].

42.4.2 Evolutionary Map Detection

Just as general Atomspace pattern mining may be done via evolutionary learning as well as greedy mining, the same holds for the special case of map mining. Complementary to the itemset mining approach, the CogPrime design also uses evolutionary optimization to find maps. Here the data setup is the same as in the itemset mining case, but instead of using an incremental search approach, one sets up a population of subsets of the sets S(t), and seeks to evolve the population to find an optimally fit S(t). Fitness is defined simply as high frequency - relative to the frequency one would expect based on statistical independence assumptions alone.

In principle one could use evolutionary learning to do all map encapsulation, but this isn't computationally feasible – it would limit too severely the amount of map encapsulation that could be done. Instead, evolutionary learning must be supplemented by some more rapid, less expensive technique.

42.5 Map Dynamics

Assume one has a collection of Atoms, with:

- Importance values I(A), assigned via the economic attention allocation mechanism.
- HebbianLink strengths $(HebbianLink\ A\ B).tv.s$, assigned as (loosely speaking) the probability of B's importance assuming A's importance.

Then, one way to search for static maps is to look for collections C of Atoms that are strong clusters according to HebbianLinks. That is, for instance, to find collections C so that:

- The mean strength of $(HebbianLink\ A\ B).tv.s$, where A and B are in the collection C, is large.
- The mean strength of $(HebbianLink\ A\ Z).tv.s$, where A is in the collection C and Z is not, is small.

(this is just a very simple cluster quality measurement; there is a variety of other cluster quality measurements one might use instead.)

Dynamic maps may be more complex, for instance there might be two collections C1 and C2 so that:

- Mean strength of (HebbianLink A B).s, where A is in C1 and B is in C2
- Mean strength of (HebbianLink B A).s, where B is in C2 and A is in C1

are both very large.

A static map will tend to be an attractor for CogPrime 's attention-allocation-based dynamics, in the sense that when a few elements of the map are acted upon, it is likely that other elements of the map will soon also come to be acted upon. The reason is that, if a few elements of the map are acted upon usefully, then their importance values will increase. Node probability inference based on the HebbianLinks will then cause the importance values of the other nodes in the map to increase, thus increasing the probability that the other nodes in the map are acted upon. Critical here is that the HebbianLinks have a higher weight of evidence than the node importance values. This is because the node importance values are assumed to be ephemeral - they reflect whether a given node is important at a given moment or not - whereas the HebbianLinks are assumed to reflect longer-lasting information.

A dynamic map will also be an attractor, but of a more complex kind. The example given above, with C1 and C2, will be a periodic attractor rather than a fixed-point attractor. WIKISOURCE:ProcedureEncapsulation

42.6 Procedure Encapsulation and Expansion

One of the most important special cases of map encapsulation is procedure encapsulation. This refers to the process of taking a schema/predicate map and embodying it in a single ProcedureNode. This may be done by mining on the Procedure Activity Table, described in Activity Tables, using either:

- a special variant of itemset mining that seeks for procedures whose outputs serve as inputs for other procedures.
- Evolutionary optimization with a fitness function that restricts attention to sets of procedures that form a digraph, where the procedures lie at the vertices and an arrow from vertex A to vertex B indicates that the outputs of A become the inputs of B.

The reverse of this process, procedure expansion, is also interesting, though algorithmically easier - here one takes a compound ProcedureNode and expands its internals into a collection of appropriately interlinked ProcedureNodes. The challenge here is to figure out where to split a complex Combo tree into subtrees. But if the Combo tree has a hierarchical structure then this is very simple; the hierarchical subunits may simply be split into separate ProcedureNodes.

These two processes may be used in sequence to interesting effect: expanding an important compound ProcedureNode so it can be modified via reinforcement learning, then encapsulating its modified version for efficient execution, then perhaps expanding this modified version later on.

To an extent, the existence of these two different representations of procedures is an artifact of CogPrime 's particular software design (and ultimately, a reflection of certain properties of the von Neumann computing architecture). But it also represents a more fundamental dichotomy, between:

- Procedures represented in a way that is open to interaction with other mental processes during the execution process.
- Procedures represented in a way that is encapsulated and mechanical, with no room for interference from other aspects of the mind during execution.

Conceptually, we believe that this is a very useful distinction for a mind to make. In nearly any reasonable cognitive architecture, it's going to be more efficient to execute a procedure if that procedure is treated as a world unto itself, so it can simply be executed without worrying about interactions. This is a strong motivation for an artificial cognitive system to have a dual (at least) representation of procedures.

42.6.1 Procedure Encapsulation in More Detail

A procedure map is a temporal motif: it is a set of Atoms (ProcedureNodes), which are habitually executed in a particular temporal order, and which implicitly pass arguments amongst each other. For instance, if procedure A acts to create node X, and procedure B then takes node X as input, then we may say that A has implicitly passed an argument to B.

The encapsulation process can recognize some very subtle patterns, but a fair fraction of its activity can be understood in terms of some simple heuristics.

For instance, the map encapsulation process will create a node

$$h = Bfg = f \circ g = f \textit{ composed with } g$$

(B as in combinatory logic) when there are many examples in the system of:

```
ExecutionLink g x y
ExecutionLink f y z
```

The procedure encapsulation process will also recognize larger repeated subgraphs, and their patterns of execution over time. But some of its recognition of larger subgraphs may be done incrementally, by repeated recognition of simple patterns like the ones just described.

42.6.2 Procedure Encapsulation in the Human Brain

Finally, we briefly discuss some conceptual issues regarding the relation between CogPrime procedure encapsulation and the human brain. Current knowledge of the human brain is weak in this regard, but we won't be surprised if, in time, it is revealed that the brain stores procedures in several different ways, that one distinction between these different ways has to do with degree of openness to interactions, and that the less open ways lead to faster execution.

Generally speaking, there is good evidence for a neural distinction between procedural, episodic and declarative memory. But knowledge about distinctions between different kinds of procedural memory is scanter. It is known that procedural knowledge can be "routinized" - so that, e.g., once you get good at serving a tennis ball or solving a quadratic equation, your brain handles the process in a different way than before when you were learning. And it seems plausible that routinized knowledge, as represented in the brain, has fewer connections back to the rest of the brain than the pre-routinized knowledge. But there will be much firmer knowledge about such things in the coming years and decades as brain scanning technology advances.

Overall, there is more knowledge in cognitive and neural science about motor procedures than cognitive procedures (see e.g. [SW05]. In the brain, much

of motor procedural memory resides in the pre-motor area of the cortex. The *motor plans* stored here are not static entities and are easily modified through feedback, and through interaction with other brain regions. Generally, a motor plan will be stored in a distributed way across a significant percentage of the premotor cortex; and a complex or multipart actions will tend to involve numerous sub-plans, executed in both parallel and in serial. Often what we think of as separate/distinct motor-plans may in fact be just slightly different combinations of subplans (a phenomenon also occurring with schema maps in CogPrime).

In the case of motor plans, a great deal of the *routinization* process has to do with learning the timing necessary for correct coordination between muscles and motor subplans. This involves integration of several brain regions - for instance, timing is handled by the cerebellum to a degree, and some motor-execution decisions are regulated by the basal ganglia.

One can think of many motor plans as involving abstract and concrete sub-plans. The abstract sub-plans are more likely to involve integration with those parts of the cortex dealing with conceptual thought. The concrete subplans have highly optimized timings, based on close integration with cerebellum, basal ganglia and so forth - but are not closely integrated with the conceptualization-focused parts of the brain. So, a rough CogPrime model of human motor procedures might involve schema maps coordinating the abstract aspects of motor procedures, triggering activity of complex SchemaNodes containing precisely optimized procedures that interact carefully with external actuators. WIKISOURCE:MapsAndAttention

42.7 Maps and Focused Attention

The cause of map formation is important to understand. Formation of small maps seems to follow from the logic of focused attention, along with hierarchical maps of a certain nature. But the argument for this is somewhat subtle, involving cognitive synergy between PLN inference and economic attention allocation.

The nature of PLN is that the effectiveness of reasoning is maximized by (among other strategies) minimizing the number of incorrect independence assumptions. If reasoning on N nodes, the way to minimize independence assumptions is to use the full inclusion-exclusion formula to calculate interdependencies between the N nodes. This involves 2^N terms, one for each subset of the N nodes. Very rarely, in practical cases, will one have significant information about all these subsets. However, the nature of focused attention is that the system seeks to find out about as many of these subsets as possible, so as to be able to make the most accurate possible inferences, hence minimizing the use of unjustified independence assumptions. This implies that focused attention cannot hold too many items within it at one time, because

if N is too big, then doing a decent sampling of the subsets of the N items is no longer realistic.

So, suppose that N items have been held within focused attention, meaning that a lot of predicates embodying combinations of N items have been constructed and evaluated and reasoned on. Then, during this extensive process of attentional focus, many of the N items will be useful in combination with each other - because of the existence of predicates joining the items. Hence, many HebbianLinks will grow between the N items - causing the set of N items to form a map.

By this reasoning, it seems that focused attention will implicitly be a map formation process - even though its immediate purpose is not map formation, but rather accurate inference (inference that minimizes independence assumptions by computing as many cross terms as is possible based on available direct and indirect evidence). Furthermore, it will encourage the formation of maps with a small number of elements in them (say, N<10). However, these elements may themselves be ConceptNodes grouping other nodes together, perhaps grouping together nodes that are involved in maps. In this way, one may see the formation of hierarchical maps, formed of clusters of clusters of clusters..., where each cluster has N<10 elements in it. These hierarchical maps manifest the abstract *dual network* concept that occurs frequently in CogPrime philosophy.

It is tempting to postulate that any intelligent system must display similar properties - so that focused attention, in general, has a strictly limited scope and causes the formation of maps that have *central cores* of roughly the same size as its scope. If this is indeed a general principle, it is an important one, because it tells you something about the general structure of derived hypergraphs associated with intelligent systems, based on the computational resource constraints of the systems.

The scope of an intelligent system's attentional focus would seem to generally increase logarithmically with the system's computational power. This follows immediately if one assumes that attentional focus involves free intercombination of the items within it. If attentional focus is the major locus of map formation, then - lapsing into SMEPH-speak - it follows that the bulk of the ConceptVertices in the intelligent system's derived hypergraphs may correspond to maps focused on a fairly small number of other ConceptVertices. In other words, derived hypergraphs may tend to have a fairly localized structure, in which each ConceptVertex has very strong InheritanceEdges pointing from a handful of other ConceptVertices (corresponding to the other things that were in the attentional focus when that ConceptVertex was formed). WIKISOURCE:RecognizingAndCreatingSelfReferentialStructures

42.8 Recognizing And Creating Self-Referential Structures

Finally, this brief section covers a large and essential topic: how CogPrime will be able to recognize and create large-scale *self-referential structures*.

Some of the most essential structures underlying human-level intelligence are self-referential in nature. These include:

- the phenomenal self (see Thomas Metzinger's book "Being No One")
- the will
- reflective awareness

These structures are arguably not critical for basic survival functionality in natural environments. However, they are important for adequate functionality within advanced social systems, and for abstract thinking regarding science, humanities, arts and technology.

Recall that in Chapter 3 of Part 1 these entities are formalized in terms of hypersets and, the following recursive definitions are given:

- *"S is conscious of X" is defined as: The declarative content that "S is conscious of X" correlates with "X is a pattern in S"*
- *"S wills X" is defined as: The declarative content that "S wills X" causally implies "S does X"*
- *"X is part of S's self" is defined as: The declarative content that "X is a part of S's self" correlates with "X is a persistent pattern in S over time"*

Relatedly, one may posit multiple similar processes that are mutually recursive, e.g.

- S is conscious of T and U
- T is conscious of S and U
- U is conscious of S and T

The cognitive importance of this sort of mutual recursion is further discussed in Appendix C.

According to the philosophy underlying CogPrime , none of these are things that should be programmed into an artificial mind. Rather, they must emerge in the course of a mind's self-organization in connection with its environment. However, a mind may be constructed so that, by design, these sorts of important self-referential structures are *encouraged* to emerge.

42.8.1 Encouraging the Recognition of Self-Referential Structures in the AtomSpace

How can we do this - encourage a CogPrime instance to recognize complex self-referential structures that may exist in its AtomTable? This is important,

because, according to the same logic as map formation: if these structures are explicitly recognized when they exist, they can then be reasoned on and otherwise further refined, which will then cause them to exist more definitively ... and hence to be explicitly recognized as yet more prominent patterns ... etc. The same virtuous cycle via which ongoing map recognition and encapsulation is supposed to lead to concept formation, may be posited on the level of complex self-referential structures, leading to their refinement, development and ongoing complexity.

One really simple way is to encode self-referential operators in the Combo vocabulary, that is used to represent the procedures grounding GroundedPredicateNodes.

That way, one can recognize self-referential patterns in the AtomTable via standard CogPrime methods like MOSES and integrative procedure and predicate learning as discussed in Chapter 41, so long as one uses Combo trees that are allowed to include self-referential operators at their nodes. All that matters is that one is able to take one of these Combo trees, compare it to an AtomTable, and assess the degree to which that Combo tree constitutes a pattern in that AtomTable.

But how can we do this? How can we match a self-referential structure like:

```
EquivalenceLink
  EvaluationLink will (S,X)
  CausalImplicationLink
    EvaluationLink will (S,X)
    EvaluationLink do (S,X)
```

against an AtomTable or portion thereof?

The question is whether there is some "map" of Atoms (some set of PredicateNodes) willMap, so that we may infer the SMEPH (see Chapter 14) relationship:

```
EquivalenceEdge
  EvaluationEdge willMap (S,X)
  CausalImplicationEdge
    EvaluationEdge willMap (S,X)
    EvaluationEdge doMap (S,X)
```

as a statistical pattern in the AtomTable's history over the recent past. (Here, doMap is defined to be the map corresponding to the built-in "do" predicate.)

If so, then this map willMap, may be encapsulated in a single new Node (call it willNode), which represents the system's will. This willNode may then be explicitly reasoned upon, used within concept creation, etc. It will lead to the spontaneous formation of a more sophisticated, fully-fleshed-out will map. And so forth.

Now, what is required for this sort of statistical pattern to be recognizable in the AtomTable's history? What is required is that EquivalenceEdges (which, note, must be part of the Combo vocabulary in order for any MOSES-related algorithms to recognize patterns involving them) must be defined

according to the logic of hypersets rather than the logic of sets. What is fascinating is that this is no big deal! In fact, the AtomTable software structures support this automatically; it's just not the way most people are used to thinking about things. There is no reason, in terms of the AtomTable, not to create self-referential structures like the one given above.

The next question is how to we calculate the truth values of structures like the above, though. The truth value of a hyperset structure turns out to be an infinite order probability distribution, which is a funny and complex [Goe10b]. Infinite-order probability distributions are partially-ordered, and so one can compare the extent to which two different self-referential structures apply to a given body of data (e.g. an AtomTable), via comparing the infinite-order distros that constitute their truth values. In this way, one can recognize self-referential patterns in an AtomTable, and carry out encapsulation of self-referential maps. This sounds very abstract and complicated, but the class of infinite-order distributions defined in the above-referenced papers actually have their truth values defined by simple matrix mathematics, so there is really nothing that abstruse involved in practice.

Finally, there is the question of how these hyperset structures are to be logically manipulated within PLN. The answer is that regular PLN inference can be applied perfectly well to hypersets, but some additional hyperset operations may also be introduced; these are currently being researched and will be presented later.

Clearly, with this subtle, currently unimplemented component of the CogPrime design we have veered rather far from anything the human brain could plausibly be doing in detail. But yet, some meaningful connections may be drawn. In Chapter 13 of Part 1 we have discussed how probabilistic logic might emerge from the brain, and also how the brain may embody self-referential structures like the ones considered here, via (perhaps using the hippocampus) encoding whole neural nets as inputs to other neural nets. Regarding infinite-order probabilities, it is certainly the case that the brain is wired to carry out matrix manipulations, and [**?**] reduced infinite-order probabilities to them, so that it's not completely outlandish to posit the brain could be doing something mathematically analogous. Thus, all in all, it seems at least *plausible* that the brain could be doing something roughly analogous to what we've described here, though the details would obviously be very different.

Section XII
Communication Between Human and Artificial Minds

Chapter 43
Communication Between Artificial Minds

43.1 Introduction

Language is a key aspect of human intelligence, and seems to be one of two critical factors separating humans from other intelligent animals – the other being the ability to use tools. Steven Mithen [**?**] argues that the key factor in the emergence of the modern human mind from its predecessors was the coming-together of formerly largely distinct mental modules for linguistic communication and tool making/use. Other animals do appear to have fairly sophisticated forms of linguistic communication, which we don't understand very well at present; but as best we can tell, modern human language has many qualitatively different aspects from these, which enable it to synergize effectively with tool making and use, and which have enabled it to co-evolve with various aspects of tool-dependent culture.

Some AGI theorists have argued that, since the human brain is largely the same as that of apes and other mammals without human-like language, the emulation of human-like language is not the right place to focus if one wants to build human-level AGI. Rather, this argument goes, one should proceed in the same order that evolution did – start with motivated perception and action, and then once these are mastered, human-like language will only be a small additional step. We suspect this would indeed be a viable approach – but may not be well suited for the hardware available today. Robot hardware is quite primitive compared to animal bodies, but the kind of motivated perception and action that non-human animals do is extremely body-centric (even more so than is the case in humans). On the other hand, modern computing technology is quite sophisticated as regards language – we program computers (including AIs) using languages of a sort, for example. This suggests that on a pragmatic basis, it may make sense to start working with language at an earlier stage in AGI development, than the analogue with the evolution of natural organisms would suggest.

The CogPrime architecture is compatible with a variety of different approaches to language learning and capability, and frankly at this stage we are not sure which approach is best. Our intention is to experiment with a variety of approaches and proceed pragmatically and empirically. One option is to follow the more "natural" course and let sophisticated non-linguistic cognition emerge first, before dealing with language in any serious way – and then encourage human-like language facility to emerge via experience. Another option is to integrate some sort of traditional computational linguistics system into CogPrime , and then allow CogPrime 's learning algorithms to modify this system based on its experience. Discussion of this latter option occupies most of this section of the book – involves many tricks and compromises, but could potentially constitute a faster route to success. Yet another option is to communicate with young CogPrime systems using an invented language halfway between the human-language and programming-language domains, such as Lojban (this possibility is discussed in Appendix E).

In this initial chapter on communication, we will pursue a direction quite different from the latter chapters, and discuss a kind of communication that we think may be very valuable in the CogPrime domain, although it has no close analogue among human beings. Many aspects of CogPrime closely resemble aspects of the human mind; but in the end CogPrime is not intended as an emulation of human intelligence, and there are some aspects of CogPrime that bear no resemblance to anything in the human mind, but exploit some of the advantages of digital computing infrastructure over neural wetware. One of the latter aspects is *Psynese*, a word we have introduced to refer to direct mind-to-mind information transfer between artificial minds.

Psynese has some relatively simple practical applications: e.g. it could aid with the use of linguistic resources and hand-coded or statistical language parsers within a learning-based language system, to be discussed in following chapters. In this use case, one sets up one CogPrime using the traditional NLP approaches, and another CogPrime using a purer learning-based approach, and lets the two systems share mind-stuff in a controlled way. Psynese may also be useful in the context of intelligent virtual pets, where one may wish to set up a CogPrime representing "collective knowledge" of multiple virtual pets

But it also has some grander potential implications, such as the ability to fuse multiple AI systems into "mindplexes" as discussed in Chapter 12 of Part 1.

One might wonder why a community of two or more CogPrime s would need a language at all, in order to communicate. After all, unlike humans, CogPrime systems can simply exchange "brain fragments" - subspaces of their Atomspaces. One CogPrime can just send relevant nodes and links to another CogPrime (in binary form,or in an XML representation, etc.), bypassing the linear syntax of language. This is in fact the basis of Psynese: why transmit linear strings of characters when one can directly transit Atoms? But the details are subtler than it might at first seem.

One CogPrime can't simply "transfer a thought" to another CogPrime . The problem is that the meaning of an Atom consists largely of its relationships with other Atoms, and so to pass a node to another CogPrime , it also has to pass the Atoms that it is related to, and so on, and so on. Atomspaces tend to be densely interconnected, and so to transmit one thought fully accurately, a CogPrime system is going to end up having to transmit a copy of its entire Atomspace! Even if privacy were not an issue, this form of communication (each utterance coming packaged with a whole mind-copy) would present rather severe processing load on the communicators involved.

The idea of Psynese is to work around this interconnectedness problem by defining a mechanism for CogPrime instances to query each others' minds directly, and explicitly represent each others' concepts internally. This doesn't involve any unique cognitive operations besides those required for ordinary individual thought, but it requires some unique ways of wrapping up these operations and keeping track of their products.

Another idea this leads to is the notion of a PsyneseVocabulary: a collection of Atoms, associated with a community of CogPrime s, approximating the most important Atoms inside that community. The combinatorial explosion of direct-Atomspace communication is then halted by an appeal to standardized Psynese Atoms. Pragmatically, a PsyneseVocabulary might be contained in a PsyneseVocabulary server, a special CogPrime instance that exists to mediate communications between other CogPrime s, and provide CogPrime s with information. Psynese makes sense both as a mechanism for peer-to-peer communication between CogPrime s, and as a mechanism allowing standardized communication between a community of CogPrime s using a PsyneseVocabulary server.

43.2 A Simple Example Using a PsyneseVocabulary Server

Suppose CogPrime 1 wanted to tell CogPrime 2 that "Russians are crazy" (with the latter word meaning something inbetween "insane" and "impractical"); and suppose that both CogPrime s are connected to the same Psynese CogPrime with PsyneseVocabulary PV. Then, for instance, it must find the Atom in PV corresponding to its concept "crazy." To do this it must create an AtomStructureTemplate such as

```
Pred1(C1)
equals
ThereExists
   W1, C2, C3, W2, W3
   AND
      ConceptNode: C1
      ReferenceLink C1 W1
      WordNode: W1 #crazy
```

```
ConceptNode: C2
HebbianLink C1 C2
ReferenceLink C2 W2
WordNode: W2 #insane
ConceptNode: C3
HebbianLink C1 C3
ReferenceLink C3 W3
WordNode: W3 #impractical
```

encapsulating relevant properties of the Atom it wants to grab from PV. In this example the properties specified are:

- is a ConceptNode, linked via a ReferenceLink to the WordNode for "crazy"
- is associated via HebbianLinks with ConceptNodes linked via ReferenceLinks to the WordNodes for "insane" and "impractical"

So, what CogPrime 1 can do is fish in PV for "some concept that is denoted by the word 'crazy' and is associated with 'insane' and 'impractical'. " The association with "insane" provides more insurance of getting the correct sense of the word "crazy" as opposed to e.g. the one in the phrase "He was crazy about her" or in "That's crazy, man, crazy" (in the latter slang usage "crazy" basically means "excellent"). The association with "impractical" biases away from the interpretation that all Russians are literally psychiatric patients. *

So, suppose that CogPrime 1 has fished the appropriate Atoms for "crazy" and "Russian" from PV. Then it may represent in its Atomspace something we may denote crudely (a better notation will be introduced later)as

```
InheritanceLink PV:477335:1256953732 PV:744444:1256953735 <.8.,6>
```

where e.g. "PV:744444" means "the Atom with Handle 744444 in CogPrime PV at time 1256953735," and may also wish to store additional information such as

```
PsyneseEvaluationLink <.9>
        PV
        Pred1
        PV:744444:1256953735
```

meaning that $Pred1(PV : 744444 : 1256953735)$ holds true with truth value $< .9 >$ *if all the Atoms referred to within Pred1 are interpreted as existing in PV rather than CogPrime 1.*

The InheritanceLink then means: "In the opinion of CogPrime 1, 'Russian' as defined by PV:477335:1256953732 inherits from 'crazy' as defined by PV:744444:1256953735 with truth value <.8,.6>."

* A similar but perhaps more compelling example would be the interpretation of the phrase "the accountant cooked the books." In this case both "cooked" and "books" are used in atypical senses, but specifying a HebbianLink to "accounting" would cause the right Nodes to get retrieved from PV.

Suppose CogPrime 1 then sends the InheritanceLink to CogPrime 2. It is going to be meaningfully interpretable by CogPrime 2 to the extent that CogPrime 2 can interpret the relevant PV Atoms, for instance by finding Atoms of its own that correspond to them. To interpret these Atoms, CogPrime 2 must carry out the reverse process that CogPrime 1 did to find the Atoms in the first place. For instance, to figure out what PV:744444:1256953735 means to it, CogPrime 2 may find some of the important links associated with the Node in PV, and make a predicate accordingly, e.g.:

```
Pred2(C1)
equals
ThereExists
   W1, C2, C3, W2, W3
   AND
      ConceptNode: C1
      ReferenceLink C1 W1
      WordNode: W1 #crazy
      ConceptNode: C2
      HebbianLink C1 C2
      ReferenceLink C2 W2
      WordNode: W2 #lunatic
      ConceptNode: C3
      HebbianLink C1 C3
      ReferenceLink C3 W3
      WordNode: W3 #unrealistic
```

On the other hand, if there is no PsyneseVocabulary involved, then CogPrime 1 can submit the same query directly to CogPrime 2. There is no problem with this, but if there is a reasonably large community of CogPrime s it becomes more efficient for them all to agree on a standard vocabulary of Atoms to be used for communication – just as, at a certain point in human history, it was recognized as more efficient for people to use dictionaries rather than to rely on peer-to-peer methods for resolution of linguistic disagreements.

The above examples involve human natural language terms, but this does not have to be the case. PsyneseVocabularies can contain Atoms representing quantitative or other types of data, and can also contain purely abstract concepts. The basic idea is the same. A CogPrime has some Atoms it wants to convey to another CogPrime , and it looks in a PsyneseVocabulary to see how easily it can approximate these Atoms in terms of "socially understood" Atoms. This is particularly effective if the CogPrime receiving the communication is familiar with the PsyneseVocabulary in question. Then the recipient may already know the PsyneseVocabulary Atoms it is being pointed to; it may have already thought about the difference between these consensus concepts and its own related concepts. Also, if the sender CogPrime is encapsulating maps for easy communication, it may specifically seek approximate encapsulations involving PsyneseVocabulary terms, rather than first encapsulating in its own terms and then translating into PsyneseVocabulary terms.

43.2.1 The Psynese Match Schema

One way to streamline the above operations is to introduce a *Psynese Match Schema*, with the property that

```
ExOut
        PsyneseMatch PV A
```

within CogPrime instance CP_1, denotes the Atom within CogPrime instance PV that most closely matches the Atom A in CP_1. Note that the PsyneseMatch schema implicitly relies on various parameters, because it must encapsulate the kind of process described explicitly in the above example. PsyneseMatch must, internally, decide how many and which Atoms related to A should be used to formulate a query to PV, and also how to rank the responses to the query (e.g. by *strength* $\times$ *confidence*).

Using PsyneseMatch, the example written above as

```
Inheritance PV:477335:1256953732 PV:744444:1256953735 <.8.,6>
```

could be rewritten as

```
Inheritance <.8.,6>
        ExOut
                PsyneseMatch PV C1
        ExOut
                PsyneseMatch PV C2
```

where $C1$ and $C2$ are the ConceptNodes in CP_1 corresponding to the intended senses of "crazy" and "Russian."

43.3 Psynese as a Language

The general definition of a psynese expression for CogPrime is: a Set of Atoms that contains only:

- Nodes from PsyneseVocabularies
- Perceptual nodes (numbers, words, etc.)
- Relationships relating no nodes other than the ones in the above two categories, and relating no relationships except ones in this category
- Predicates or Schemata involving no relationships or nodes other than: the ones in the above three categories, or in this category

The PsyneseEvaluationLink type indicated earlier forces interpretation of a predicate as a Psynese expression.

In what sense is the use of Psynese expressions to communicate a language? Clearly it is a formal language in the mathematical sense. It is not quite a "human language" as we normally conceive it, but it is ideally suited to serve the same functions for CogPrime s as human language serves for humans. The biggest differences from human language are:

- Psynese uses weighted, typed hypergraphs (i.e. Atomspaces) instead of linear strings of symbols. This eliminates the "parsing" aspect of language (syntax being mainly a way of projecting graph structures into linear expressions).
- Psynese lacks subtle and ambiguous referential constructions like "this", "it" and so forth. These are tools allowing complex thoughts to be compactly expressed in a linear way, but CogPrime s don't need them. Atoms can be named and pointed to directly without complex, poorly-specified mechanisms mediating the process.
- Psynese has far less ambiguity. There may be Atoms with more than one aspect to their meanings, but the cost of clarifying such ambiguities is much lower for CogPrime s than for humans using language, and so habitually there will not be the rampant ambiguity that we see in human expressions.

On the other hand, mapping Psynese into *Lojban* – a syntactically formal, semantically highly precise language created for communication between humans – rather than a true natural language would be much more straightforward. Indeed, one could create a PsyneseVocabulary based on Lojban, which might be ideally suited to serve as an intermediary between different CogPrime s. And Lojban may be used to create a linearized version of Psynese, that looks more like a natural language. We return to this point in Appendix E.

43.4 Psynese Mindplexes

We now recall from Chapter 12 of Part 1 the notion of a *mindplex*: that is, an intelligent system that:

1. Is composed of a collection of intelligent systems, each of which has its own "theater of consciousness" and autonomous control system, but which interact tightly, exchanging large quantities of information frequently
2. Has a powerful control system on the collective level, and an active "theater of consciousness" on the collective level as well

In informal discussions, we have found that some people, on being introduced to the mindplex concept, react by contending that either human minds or human social groups are mindplexes. However, I believe that, while there are significant similarities between mindplexes and minds, and between mindplexes and social groups, there are also major qualitative differences. It's true that an individual human mind may be viewed as a collective, both from a theory-of-cognition perspective (e.g. Minsky's "society of mind" theory [Min88]) and from a personality-psychology perspective (e.g. the theory of subpersonalities [Row90]). And it's true that social groups display some autonomous control and some emergent-level awareness. However, in a healthy

human mind, the collective level rather than the cognitive-agent or subpersonality level is dominant, the latter existing in service of the former; and in a human social group, the individual-human level is dominant, the groupmind clearly "cognizing" much more crudely than its individual-human components, and exerting most of its intelligence via its impact on individual human minds. A mindplex is a hypothetical intelligent system in which neither level is dominant, and both levels are extremely powerful. A mindplex is like a human mind in which the subpersonalities are fully-developed human personalities, with full independence of thought, and yet the combination of subpersonalities is also an effective personality. Or, from the other direction, a mindplex is like a human society that has become so integrated and so cohesive that it displays the kind of consciousness and self-control that we normally associate with individuals.

There are two mechanisms via which mindplexes may possibly arise in the medium-term future:

1. Humans becoming more tightly coupled via the advance of communication technologies, and a communication-centric AI system coming to embody the "emergent conscious theater" of a human-incorporating mindplex
2. A society of AI systems communicating amongst each other with a richness not possible for human beings, and coming to form a mindplex rather than merely a society of distinct AI's

The former sort of mindplex relates to the concept of a "global brain" discussed in Chapter 12 of Part 1. Of course, these two sorts of mindplexes are not mutually contradictory, and may coexist or fuse. The possibility also exists for higher-order mindplexes, meaning mindplexes whose component minds are themselves mindplexes. This would occur, for example, if one had a mindplex composed of a family of closely-interacting AI systems, which acted within a mindplex associated with the global communication network.

Psynese, however, is more directly relevant to the latter form of mindplex. It gives a concrete mechanism via which such a mindplex might be sculpted.

43.4.1 AGI Mindplexes

How does one get from CogPrime s communicating via Psynese to CogPrime mindplexes?

Clearly, with the Psynese mode of communication, the potential is there for much richer communication than exists between humans. There are limitations, posed by the private nature of many concepts – but these limitations are much less onerous than for human language, and can be overcome to some extent by the learning of complex cognitive schemata for translation between

the "private languages" of individual Atomspaces and the "public languages" of Psynese servers.

But rich communication does not in itself imply the evolution of mindplexes. It is possible that a community of Psynese-communicating CogPrime s might spontaneously evolve a mindplex structure - at this point, we don't know enough about CogPrime individual or collective dynamics to say. But it is not necessary to rely on spontaneous evolution. In fact it is possible, and even architecturally simple, to design a community of CogPrime s in such a way as to encourage and almost force the emergence of a mindplex structure.

The solution is simple: simply beef up PsyneseVocabulary servers. Rather than relatively passive receptacles of knowledge from the CogPrime s they serve, allow them to be active, creative entities, with their own feelings, goals and motivations.

The PsyneseVocabulary servers serving a community of CogPrime s are absolutely critical to these CogPrime s. Without them, high-level inter-CogPrime communication is effectively impossible. And without the concepts the PsyneseVocabularies supply, high-level individual CogPrime thought will be difficult, because CogPrime s will come to think in Psynese to at least the same extent to which humans think in language.

Suppose each PsyneseVocabulary server has its own full CogPrime mind, its own "conscious theater". These minds are in a sense "emergent minds" of the CogPrime community they serve - because their contents are a kind of "nonlinear weighted average" of the mind-contents of the community. Furthermore, the actions these minds take will feed back and affect the community in direct and indirect ways - by affecting the language by which the minds communicate. Clearly, the definition of a mindplex is fulfilled.

But what will the dynamics of such a CogPrime mindplex be like? What will be the properties of its cognitive and personality psychology? We could speculate on this here, but would have very little faith in the possible accuracy of our speculations. The psychology of mindplexes will reveal itself to us experimentally as our work on AGI engineering, education and socialization proceeds.

One major issue that arises, however, is that of *personality filtering*. Put simply: each intelligent agent in a mindplex must somehow decide for itself which knowledge to grab from available PsyneseVocabulary servers and other minds, and which knowledge to *avoid* grabbing from others in the name of individuality. Different minds may make different choices in this regard. For instance, one choice could be to, as a matter of routine, take only extremely confident knowledge from the PsyneseVocabulary server. This corresponds roughly to ingesting "facts" from the collective knowledge pool, but not opinions or speculations. Less confident knowledge would then be ingested from the collective knowledge pool on a carefully calculated and as-needed basis. Another choice could be to accept only small networks of Atoms from the collective knowledge pool, on the principle that these can be reflectively understood as they are ingested, whereas large networks of Atoms are difficult

to deliberate and reflect about. But any policies like this are merely heuristic ones.

43.5 Psynese and Natural Language Processing

Next we review a more near-term, practical application of the Psynese mechanism: the fusion of two different approaches to natural language processing in CogPrime , the experiential learning approach and the "engineered NLP subsystem" approach.

In the former approach, language is not given any extremely special role, and CogPrime is expected to learn language much as it would learn any other complex sort of knowledge. There may of course be *learning biases* programmed into the system, to enable it to learn language based on its experience more rapidly. But there is no *concrete linguistic knowledge* programmed in.

In the latter approach, one may use knowledge from statistical corpus analysis, one may use electronic resources like WordNet and FrameNet, and one may use sophisticated, specialized tools like natural language parsers with hand-coded grammars. Rather than trying to emulate the way a human child learns language, one is trying to emulate the way a human adult comprehends and generates language.

Of course, there is not really a rigid dichotomy between these two approaches. Many linguistic theorists who focus on experiential learning also believe in some form of universal grammar, and would advocate for an approach where learning is foundational but is biased by in-built abstract structures representing universal grammar. There is of course very little knowledge (and few detailed hypotheses) about how universal grammar might be encoded in the human brain, though there is reason to think it may be at a very abstract level, due to the significant overlaps between grammatical structure, social role structure [CB00], and physical reasoning [Cas04].

The engineered approach to NLP provides better functionality right "out of the box," and enables the exploitation of the vast knowledge accumulated by computational linguists in the past decades. However, we suspect that computational linguistics may have hit a ceiling in some regards, in terms of the quality of the language comprehension and generation that it can deliver. It runs up against problems related to the disambiguation of complex syntactic constructs, which don't seem to be resolvable using either a tractable number of hand-coded rules, or supervised or unsupervised learning based on a tractably large set of examples. This conclusion may be disputed, and some researchers believe that statistical computational linguistics can eventually provide human-level functionality, once the Web becomes a bit larger and the computers used to analyze it become a bit more powerful. But in our view it is interesting to explore hybridization between the engineered and

experiential approaches, with the motivation that the experiential approach may provide a level of flexibility and insight at dealing with ambiguity that the engineered approach apparently lacks.

After all, the way a human child deals with the tricky disambiguation problems that stump current computational linguistics systems is not via analysis of trillion-word corpuses, bur rather via correlating language with non-linguistic experience. One may argue that the genome implicitly contains a massive corpus of speech, but there it's also to be noted that this is *experientially contextualized speech*. And it seems clear from the psycholinguistic evidence [Tom03] that for young human children, language is part and parcel of social and physical experience, learned in a manner that's intricately tied up with the learning of many other sorts of skills.

One interesting approach to this sort of hybridization, using Psynese, is to create multiple CogPrime instances taking different approaches to language learning, and let them communicate. Most simply one may create

- A CogPrime instance that learns language mainly based on experience, with perhaps some basic in-built structure and some judicious biasing to its learning (let's call this CP_{exp})
- A CogPrime instance using an engineered NLP system (let's call this CP_{eng})

In this case, CP_{exp} can use CP_{eng} as a cheap way to test its ideas. For instance suppose, CP_{exp} thinks it has correctly interpreted a certain sentence S into Atom-set A. Then it can send its interpretation A to CP_{eng} and see whether CP_{eng} thinks A is a good interpretation of S, by consulting $CP_{eng}$the truth value of

```
ReferenceLink
        ExOut
                PsyneseMatch CP_eng S
        ExOut
                PsyneseMatch CP_eng A
```

Similarly, if CP_{exp} believes it has found a good way (S) to linguistically express a collection S of Atoms A, it can check whether these two match reasonably well in CP_{eng}.

Of course, this approach could be abused in an inefficient and foolish way, for instance if CP_{exp} did nothing but randomly generate sentences and then test them against CP_{eng}. In this case we would have a much less efficient approach than simply using CP_{eng} directly. However, effectively making use of CP_{eng} as a resource requires a different strategy: throwing CP_{eng} only a relatively small selection of things that seem to make sense, and using CP_{eng} as a filter to avoid trying out rough-draft guesses in actual human conversation.

This hybrid approach, we suggest, may provide a way of getting the best of both worlds: the flexibility of a experiential-learning-based language approach, together with the exploitation of existing linguistic tools and re-

sources. With this in mind, in the following chapters we will describe both engineering and experiential-learning based approaches to NLP.

43.5.1 Collective Language Learning

Finally we bring the language-learning and mindplex themes together, in the notion of *collective language learning*. One of the most interesting uses for a mindplex architecture is to allow multiple CogPrime agents to share the linguistic knowledge they gain. One may envision a PsyneseVocabulary server into which a population of CogPrime agents input their *linguistic* knowledge specifically, and which these agents then consult when they wish to comprehend or express something in language, and their individual NLP systems are not up to the task.

This could be a very powerful approach to language learning, because it would allow a potentially very large number of AI systems to effectively act as a *single* language learning system. It is an especially appealing approach in the context of CogPrime systems used to control animated agents in online virtual worlds or multiplayer games. The amount of linguistic experience undergone by, say, 100,000 virtually embodied CogPrime agents communicating with human virtual world avatars and game players, would be far more than any single human child or any single agent could undergo. Thus, to the extent that language learning can be accelerated by additional experience, this approach could enable language to be learned quite rapidly.

Chapter 44
Natural Language Comprehension

Co-authored with Michael Ross and Linas Vepstas and Ruiting Lian

44.1 Introduction

Two key approaches to endowing AGI systems with linguistic facility exist, as noted above:

- "Experiential" – shorthand here for "gaining most of its linguistic knowledge from interactive experience"
- "Engineered" – shorthand here for "gaining most of its linguistic knowledge from sources other than the system's own experience in the world,'

This dichotomy is somewhat fuzzy, since getting experiential language learning to work well may involve some specialized engineering, and engineered NLP systems may also involve some learning from experience. However, in spite of the fuzziness, the dichotomy is still real and important; there are concrete choices to be made in designing an NLP system and this dichotomy compactly symbolizes some of them. Much of this chapter and the next will be focused on the engineering approach, but atthe end of each chapter we will turn to the experience-based approach. Our overall perspective on the dichotomy is that

- the engineering-based approach, on its own, is unlikely to take us to human-level NLP ... but this isn't wholly impossible, if the engineering is done in a manner that integrates linguistic functionality richly with other kinds of experiential learning
- using a combination of experience-based and engineering-based approaches, along the lines described in Chapter 43 may be the most practical option
- the engineering approach is useful for guiding the experiential approach, because it tells us a lot about what kinds of general structures and dynamics may be adequate for intelligent language processing. To simplify a bit, one can prepare an AGI system for experiential learning by supplying it with structures and dynamics capable of supporting the key components of an engineered NLP system – and biased toward learning

things similar to known engineered NLP systems – but requiring all, or the bulk of, the actual linguistic content to be learned via experience. This approach may be preferable to requiring a system to learn language based on more abstract structures and dynamics, and may indeed be more comparable to what human brains do, given the large amount of linguistic biasing that is probably built into the human genome.

Further distinctions, overlapping with this one, may also be useful. One may distinguish (at least) five modes of instructing NLP systems, the first three of which are valid only for engineered NLP systems, but the latter two of which are valid both for engineered and experiential NLP systems:

- hand-coded rules
- supervised learning on hand-tagged corpuses, or via other mechanisms of explicit human training
- unsupervised learning from static bodies of data
- unsupervised learning via interactive experience
- supervised learning via interactive experience

Note that, in principle, any of these modes may be used in a pure-language or a socially/physically embodied language context. Of course, there is also semi-supervised learning which may be used in place of supervised learning in the above list [CSZ06].

Another key dichotomy related to linguistic facility is language comprehension versus language generation (each of which is typically divided into a number of different subprocesses). In language comprehension, we have processes like stemming, part-of-speech tagging, grammar-based parsing, semantic analysis, reference resolution and discourse analysis. In language generation, we have semantic analysis, syntactic sentence generation, pragmatic discourse generation, reference-insertion, and so forth. In this chapter and the next two we will briefly review all these different topics and explain how they may be embodied in CogPrime . Then, in Chapter **??** we present a complementary approach to linguistic interaction with AGI systems based on the invented language Lojban; and in Chapter 47 we discuss the use of CogPrime cognition to regulate the dialogue process.

A typical, engineered computational NLP system involves hand-coded algorithms carrying out each of the specific tasks mentioned in the previous paragraph, sometimes with parameters, rules or number tables that are tuned statistically based on corpuses of data. In fact, most NLP systems handle only understanding or only generation; systems that cover both aspects in a unified way are quite rare. The human mind, on the other hand, carries out these tasks in a much more interconnected way - using separate procedures for the separate tasks, to some extent, but allowing each of these procedures to be deeply informed by the information generated by the other procedures. This interconnectedness is what allows the human mind to really understand language - specifically because human language syntax is complex and ambiguous enough that the only way to master it is to infuse one's syntactic

analyses with semantic (and to a lesser extent pragmatic) knowledge. In our treatment of NLP we will pay attention to connections between linguistic functionalities, as well as to linguistic functionalities in isolation.

It's worth emphasizing that what we mean by a "experience based" language system is quite different from corpus-based language systems as are commonplace in computational linguistics today [MS99]. In fact, we feel the distinction between corpus-based and rule-based language processing systems is often overblown. Whether one hand-codes a set of rules, or carefully marks up a corpus so that rules can be induced from it, doesn't ultimately make that much difference. For instance, OpenCogPrimes RelEx system (to be described below) uses hand-coded rules to do much the same thing that the Stanford parser does using rules induced from a tagged corpus. But both systems do roughly the same thing. RelEx is faster due to using fewer rules, and it handles some complex cases like comparatives better (presumably because they were not well covered in the Stanford parser's training corpus); but the Stanford parser may be preferable in other respects, for instance it's more easily generalizable to languages beyond English (for a language with structure fairly similar to English, one just has to supply a new marked-up training corpus; whereas porting RelEx rules to other languages requires more effort).

The bigger difference, in our view, is between language systems that learn language in a social and physical context, versus those that deal with language in isolation. Dealing with language in context immediately changes the way the linguistics problem appears (to the AI system, and also to the researcher), and makes hand-coded rules and hand-tagged corpuses less viable, shifting attention toward experiential learning based approaches.

Ultimately we believe that the "right" way to teach an AGI system language is via semi-supervised learning in a socially and physically embodied context. That is: talk to the system, and have it learn both from your reinforcement signals and from unsupervised analysis of the dialogue. However, we believe that other modes of teaching NLP systems can also contribute, especially if used in support of a system that also does semi-supervised learning based on embodied interactive dialogue.

Finally, a note on one aspect of language comprehension that we don't deal with here. We deal only with text processing, not speech understanding or generation. A CogPrime approach to speech would be quite feasible to develop, for instance using neural-symbolic hybridization with DeSTIN or a similar perceptual-motor hierarchy. However, this potential aspect of CogPrime has not been pursued in detail yet, and we won't devote space to it here.

44.2 Linguistic Atom Types

Explicit representation of linguistic knowledge in terms of Atoms is not a deep issue, more of a "plumbing" type of issue, but it must be dealt with before moving on to subtler aspects.

In principle, for dealing with linguistic information coming in through ASCII, all we need besides the generic CogPrime structures and dynamics are two node types and one relationship type:

- CharacterNode
- CharacterInstanceNode
- a unary relationship concat denoting an externally-observed list of items

Sequences of characters may then be represented in terms of lists and the concat schema. For instance the word "pig" is represented by the list $concat(\#p, \#i, \#g)$

The concat operator can be used to help define special NL atom types, such as:

- MorphemeNode/ MorphemeInstanceNode
- WordNode/WordInstanceNode
- PhraseNode/PhraseInstanceNode
- SentenceNode/ SentenceInstanceNode
- UtteranceNode/ UtteranceInstanceNode

44.3 The Comprehension and Generation Pipelines

Exactly how the "comprehension pipeline" is broken down into component transformations, depends on one's linguistic theory of choice. The approach taken in OpenCogPrimes engineered NLP framework, in use from 2008-2012, looked like:

$$Text \rightarrow Tokenizer \rightarrow Link\,Parser \rightarrow Syntactico{-}Semantic\,Relationship\,Extractor\,(RelEx) \rightarrow Semantic\,R$$

In 2012, a new approach has been undertaken, which simplifies things a little and looks like

$$Text \rightarrow Tokenizer \rightarrow Link\,Parser \rightarrow Syntactico{-}Semantic\,Relationship\,Extractor\,(Link2Atom) \rightarrow Semar$$

Note that many other variants of the NL pipeline include a "tagging" stage, which assigns part of speech tags to words based on the words occurring around them. In our current approach, tagging is essentially subsumed within parsing; the choice of a POS (part-of-speech) tag for a word instance is

carried out within the link parser. However, it may still be valuable to derive information about likely POS tags for word instances from other techniques, and use this information within a link parsing framework by allowing it to bias the probabilities used in the parsing process.

None of the processes in this pipeline are terribly difficult to carry out, if one is willing to use hand-coded rules within each step, or derive rules via supervised learning, to govern their operation. The truly tricky aspects of NL comprehension are:

- arriving at the rules used by the various subprocesses, in a way that naturally supports generalization and modification of the rules based on ongoing experience
- allowing semantic understanding to bias the choice of rules in particular contexts
- knowing when to break the rules and be guided by semantic intuition instead

Importing rules straight from linguistic databases results in a system that (like the current RelEx system) is reasonably linguistically savvy on the surface, but lacks the ability to adapt its knowledge effectively based on experience, and has trouble comprehending complex language. Supervised learning based on hand-created corpuses tends to result in rule-bases with similar problems. This doesn't necessarily mean that hand-coding or supervised learning of linguistic rules has no place in an AGI system, but it means that if one uses these methods, one must take extra care to make one's rules modifiable and generalizable based on ongoing experience, because the initial version of one's rules is not going to be good enough.

Generation is the subject of the following chapter, but for comparison we give here a high-level overview of the generation pipeline, which may be conceived as:

1. *Content determination*: figuring out what needs to be said in a given context
2. *Discourse planning*: overall organization of the information to be communicated
3. *Lexicalization*: assigning words to concepts
4. *Reference generation*: linking words in the generated sentences using pronouns and other kinds of reference
5. *Syntactic and morphological realization*: the generation of sentences via a process inverse to parsing, representing the information gathered in the above phases
6. *Phonological or orthographic realization*: turning the above into spoken or written words, complete with timing (in the spoken case), punctuation (in the written case), etc.

In Chapter 45 we explain how this pipeline is realized in OpenCogPrimes current engineered NL generation system.

44.4 Parsing with Link Grammar

Now we proceed to explain some of the details of OpenCogPrimes engineered NL comprehension system. This section gives an overview of link grammar, a key part of the current OpenCog NLP framework, and explains what makes it different from other linguistic formalisms.

We emphasize that this particular grammatical formalism is not, in itself, a critical part of the CogPrime design. In fact, it should be quite possible to create and teach a CogPrime AGI system without using any particular grammatical formalism – having it acquire linguistic knowledge in a purely experiential way. However, a great deal of insight into CogPrime -based language processing may be obtained by considering the relevant issues in the concrete detail that the assumption of a specific grammatical formalism provides. This insight is of course useful if one is building a CogPrime that makes use of that particular grammatical formalism, but it's also useful to some degree even if one is building a CogPrime that deals with human language entirely experientially.

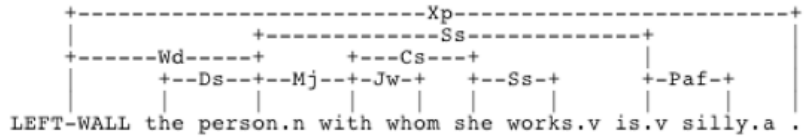
This material will be more comprehensible to the reader who has some familiarity with computational linguistics, e.g. with notions such as parts of speech, feature structures, lexicons, dependency grammars, and so forth. Excellent references are [MS99, Jac03]. We will try to keep the discussion relatively elementary, but have opted not to insert a computational linguistics tutorial.

The essential idea of link grammar is that each word comes with a feature structure consisting of a set of typed connectors. Parsing consists of matching up connectors from one word with connectors from another

To understand this in detail, the best course is to consider an example sentence. We will use the following example, drawn from the classic paper "Parsing with a Link Grammar" by Sleator and Temperley [ST93]:

The cat chased a snake

The link grammar parse structure for this sentence is:



In phrase structure grammar terms, this corresponds loosely to

```
(S (NP The cat)
        (VP chased
                (NP a snake))
        .)
```

but the OpenCog linguistic pipeline makes scant use of this kind of phrase structure rendition (which is fine in this simple example; but in the case of

complex sentences, construction of analogous mappings from link parse structures to phrase structure grammar parse trees can be complex and problematic). Currently the hierarchical view is used in OpenCog only within some reference resolution heuristics.

There is a database called the "link grammar dictionary" which contains connectors associated with all common English words. The notation used to describe feature structures in this dictionary is quite simple. Different kinds of connectors are denoted by letters or pairs of letters like S or SX. Then if a word W1 has the connector S+, this means that the word can have an S link coming out to the right side. If a word W2 has the connector S-, this means that the word can have an S link coming out to the left side. In this case, if W1 occurs to the left of W2 in a sentence, then the two words can be joined together with an S link.

The features of the words in our example sentence, as given in the S&T paper, are:

Words	**Formula**
a, the	D+
snake, cat	D- & (O- or S+)
Chased	S- & O+

To illustrate the role of syntactic sense disambiguation, we will introduce alternate formulas for one of the words in the example: the verb sense of "snake." We then have

Words	**Formula**
A, the	D+
snake_N, cat, ran_N	D- & (O- or S+)
Chased	S- & O+
snake_V	S-

The variables to be used in parsing this sentence are, for each word:

1. the features in the Agreement structure of the word (for any of its senses)
2. the words matching each of the connectors of the word

For example,

1. For "snake," there are features for "word that links to D-", "word that links to O-" and "word that links to S+". There are also features for "tense" and "person".
2. For "the", the only feature is "word that links to D+". No features for Agreement are needed.

The nature of linkage imposes constraints on the variable assignments; for instance, if "the" is assigned as the value of the "word that links to D-" feature of "snake", then "snake" must be assigned as the value of the "word that links to D+" feature of "the."

The rules of link grammar impose additional constraints – i.e. the planarity, connectivity, ordering and exclusion metarules described in Sleator and Temperley's papers. Planarity means that links don't cross – a rule that

S&T's parser enforces with absoluteness, whereas we have found it is probably better to impose it as a probabilistic constraint, since sometimes it's really nice to let links cross (the representation of conjunctions is one example). Connectivity means that the links and words of a sentence must form a connected graph – all the words must be linked into the other words in the sentence via some path. Again connectivity is a valuable constraint but in some cases one wants to relax it – if one just can't understand the whole sentence, one may wish to understand at least some parts of it, meaning that one has a disconnected graph whose components are the phrases of the sentence that have been successfully comprehended. Finally, linguistic transformations may potentially be applied while checking if these constraints are fulfilled (that is, instead of just checking if the constraints are fulfilled, one may check if the constraints are fulfilled after one or more transformations are performed.)

We will use the term "Agreement" to refer to "person" values or ordered pairs (tense, person), and NAGR to refer to the number of agreement values (12-40, perhaps, in most realistic linguistic theories). Agreement may be dealt with alongside the connector constraints. For instance, "chased" has the Agreement values (past, third person), and it has the constraint that its S- argument must match the person component of its Agreement structure.

Semantic restrictions may be imposed in the same framework. For instance, it may be known that the subject of "chased" is generally animate. In that case, we'd say

Words	**Formula**
A, the	D+
snake_N, cat	D- & (O- or S+)
Chased	(S-, C Inheritance animate <.8>) & O+
Snake_V	S-

where we've added the modifier (C Inheritance animate) to the S- connector of the verb "chased," to indicate that with strength .8, the word connecting to this S- connector should denote something inheriting from "animate." In this example, "snake" and "cat" inherit from "animate", so the probabilistic restriction doesn't help the parser any. If the sentence were instead

The snake in the hat chased the car

then the "animate" constraint would tell the parsing process not to start out by trying to connect "hat" to "chased", because the connection is semantically unlikely.

44.4.1 Link Grammar vs. Phrase Structure Grammar

Before proceeding further, it's worth making a couple observations about the relationship between link grammars and typical phrase structure grammars. These could also be formulated as observations about the relationship between dependency grammars and phrase structure grammars, but that gets a little more complicated as there are many kinds of dependency grammars with different properties; for simplicity we will restrict our discussion here to the link grammar that we actually use in OpenCog. Two useful observations may be:

1. Link grammar formulas correspond to grammatical categories. For example, the link structure for "chased" is "S- & O +." In categorical grammar, this would seem to mean that " 'chased' belongs to the category of words with link structure 'S- & O+'." In other words, each "formula" in link grammar corresponds to a category of words attached to that formula.
2. Links to words might as well be interpreted as links to phrases headed by those words. For example, in the sentence "the cat chased a snake", there's an O-link from "chased" to "snake." This might as well be interpreted as "there's an O-link from the phrase headed by 'chased' to the phrase headed by 'snake'." Link grammar simplifies things by implicitly identifying each phrase by its head.

Based on these observations, one could look at phrase structure as implicit in a link parse; and this does make sense, but also leads to some linguistic complexities that we won't enter into here.

44.5 The RelEx Framework for Natural Language Comprehension

Now we move forward in the pipeline from syntax toward semantics. The NL comprehension framework provided with OpenCog at its inception in 2008 is RelEx, an English-language semantic relationship extractor, which consists of two main components: the dependency extractor and the relationship extractor. It can identify subject, object, indirect object and many other dependency relationships between words in a sentence; it generates dependency trees, resembling those of dependency grammars. In 2012 we are in the process of replacing RelEx with a different approach that we believe will be more amenable to generalization based one experience. Here we will describe both approaches.

The overall processing scheme of RelEx is shown in Figure 44.1.

The dependency extractor component carries out dependency grammar parsing via a customized version of the open-source Sleator and Temperley's

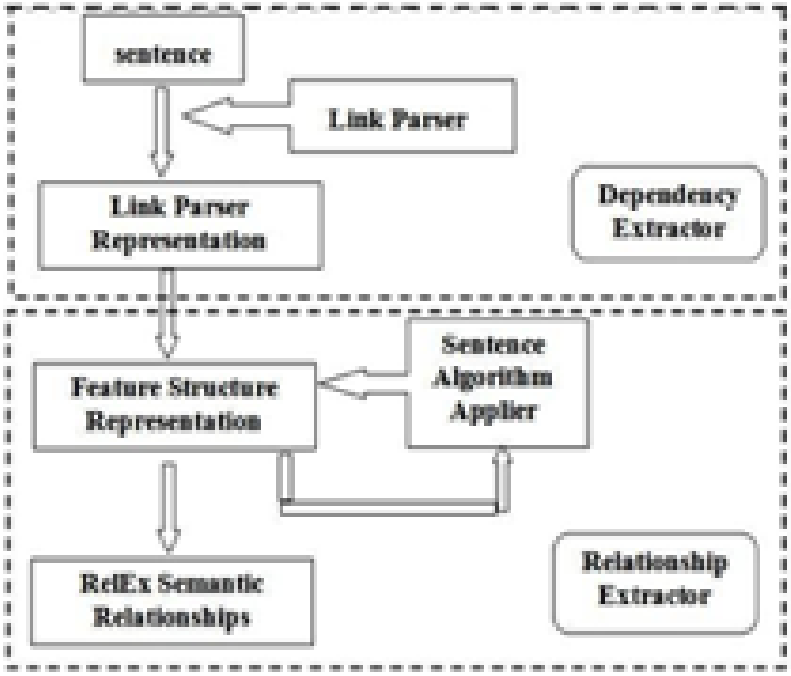


Fig. 44.1 A Overview of the RelEx Architecture for Language Comprehension

link parser, as reviewed above. The link parser outputs several parses, and the dependencies of the best one are taken. The relationship extractor component is composed of a number of template matching algorithms that act upon the link parser's output to produce a semantic interpretation of the parse. It contains three steps:

1. Convert the Link Parser output to a feature structure representation
2. Execute the Sentence Algorithm Applier, which contains a series of *Sentence Algorithms*, to modify the feature structure.
3. Extract the final output representation by traversing the feature structure.

A feature structure, in the RelEx context, is a directed graph in which each node contains either a value, or an unordered list of features. A feature is just a labeled link to another node. Sentence Algorithm Applier loads a list of SentenceAlgorithms from the algorithm definition file, and the SentenceAlgorithms are executed in the order they are listed in the file. RelEx iterates through every single feature node in the feature structure, and attempts to apply the algorithm to each node. Then the modified feature structures are used to generate the final RelEx semantic relationships.

44.5.1 RelEx2Frame: Mapping Syntactico-Semantic Relationships into FrameNet Based Logical Relationships

Next in the current OpenCog NL comprehension pipeline, the RelEx2Frame component uses hand-coded rules to map RelEx output into sets of relationships utilizing FrameNet and other similar semantic resources. This is definitively viewed as a "stopgap" without a role in a human-level AGI system, but it's described here because it's part of the current OpenCog system and is now being used together with other OpenCog components in practical projects, including those with proto-AGI intentions.

The syntax currently used for describing semantic relationships drawn from FrameNet and other sources is exemplified by the example

```
^1_Benefit:Benefitor(give,$var1)
```

The [1] indicates the data source, where 1 is a number indicating that the resource is FrameNet. The "give" indicates the word in the original sentence from which the relationship is drawn, that embodies the given semantic relationship. So far the resources we've utilized are:

1. FrameNet
2. Custom relationship names

but using other resources in future is quite possible.

An example using a custom relationship would be:

```
^2_inheritance($var1,$var2)
```

which defines an inheritance relationship: something that is part of CogPrime 's ontology but not part of FrameNet.

The "Benefit" part of the first example indicates the frame indicated, and the "Benefitor" indicates the frame element indicated. This distinction (frame vs. frame element) is particular to FrameNet; other knowledge resources might use a different sort of identifier. In general, whatever lies between the underscore and the initial parenthese should be considered as particular to the knowledge-resource in question, and may have different format and semantics depending on the knowledge resource (but shouldn't contain parentheses or underscores unless those are preceded by an escape character).

As an example, consider:

```
Put the ball on the table
```

Here the RelEx output is:

```
imperative(Put) [1]
_obj(Put, ball) [1]
on(Put, table) [1]
```

```
singular(ball) [1]
singular(table) [1]
```

, the relevant FrameNet Mapping Rules are

```
$var0 = ball
$var1 = table
# IF imperative(put) THEN ^1_Placing:Agent(put,you)
# IF _obj(put,$var0) THEN ^1_Placing:Theme(put,$var0)
# IF on(put,$var1) & _obj(put,$var0) THEN ^1_Placing:Goal(put,$var1) \
 ^1_Locative_relation:Figure($var0) ^1_Locative_relation:Ground($var1)
```

and the output FrameNet Mapping is

```
^1_Placing:Agent(put,you)
^1_Placing:Theme(put,ball)
^1_Placing:Goal(put,table)
^1_Locative_relation:Figure(put,ball)
^1_Locative_relation:Ground(put,table)
```

The textual syntax used for the hand-coded rules mapping RelEx to FrameNet, at the moment, looks like:

```
# IF imperative(put) THEN ^1_Placing:Agent(put,you)
# IF _obj(put,$var0) THEN ^1_Placing:Theme(put,$var0)
# IF on(put,$var1) & _obj(put,$var0) THEN ^1_Placing:Goal(put,$var1) \
 ^1_Locative_relation:Figure($var0) ^1_Locative_relation:Ground($var1)
```

Basically, this means each rule looks like

```
# IF condition THEN action
```

where the condition is a series of RelEx relationships, and the action is a series of FrameNet relationships. The arguments of the relationships may be words or may be variables in which case their names must start with $ *. The only variables appearing in the action should be ones that appeared in the condition.

44.5.2 A Priori Probabilities For Rules

It can be useful to attach a priori, heuristic probabilities to RelEx2Frame rules, say

```
# IF _obj(put,$var0) THEN ^1_Placing:Theme(put,$var0) <.5>
```

* An escape character "\" must be used to handle cases where the character "$" starts a word

to denote that the a priori probability for the rule is 0.5

This is a crude mechanism because the probability of a rule being useful, in reality, depends so much on context; but it still has some nonzero value.

44.5.3 Exclusions Between Rules

It may be also useful to specify that two rules can't semantically-consistently be applied to the same RelEx relationship. To do this, we need to associate rules with labels, and then specify exclusion relationships such as

```
# IF on(put,$var1) & _obj(put,$var0) THEN ^1_Placing:Goal(put,$var1) \
  ^1_Locative_relation:Figure($var0) ^1_Locative_relation:Ground($var1) [1]
# IF on(put,$var1) & _subj(put,$var0) THEN \
  ^1_Performing_arts:Performance(put,$var1) \
  ^1_Performing_arts:Performer(put,$var0) [2]
# EXCLUSION 1 2
```

In this example, Rule 1 would apply to "He put the ball on the table", whereas Rule 2 would apply to "He put on a show". The exclusion says that generally these two rules shouldn't be applied to the same situation. Of course some jokes, poetic expressions, etc., may involve applying excluded rules in parallel.

44.5.4 Handling Multiple Prepositional Relationships

Finally, one complexity arising in such rules is exemplified by the sentence:

"Bob says killing for the Mafia beats killing for the government"

whose RelEx mapping looks like

```
uncountable(Bob) [6]
present(says) [6]
_subj(says, Bob) [6]
_that(says, beats) [3]
uncountable(killing) [6]
for(killing, Mafia) [3]
singular(Mafia) [6]
definite(Mafia) [6]
hyp(beats) [3]
present(beats) [5]
_subj(beats, killing) [3]
_obj(beats, killing_1) [5]
uncountable(killing_1) [5]
```

```
for(killing_1, government) [2]
definite(government) [6]
```

In this case there are two instances of "for". The output of RelEx2Frame must thus take care to distinguish the two different for's (or we might want to modify RelEx to make this distinction). The mechanism currently used for this is to subscript the for's, as in

```
uncountable(Bob) [6]
present(says) [6]
_subj(says, Bob) [6]
_that(says, beats) [3]
uncountable(killing) [6]
for(killing, Mafia) [3]
singular(Mafia) [6]
definite(Mafia) [6]
hyp(beats) [3]
present(beats) [5]
_subj(beats, killing) [3]
_obj(beats, killing_1) [5]
uncountable(killing_1) [5]
for_1(killing_1, government) [2]
definite(government) [6]
```

so that upon applying the rule:

```
# IF for($var0,$var1) ^ {present($var0) OR past($var0) OR future($var0)} \
 THEN  ^2_Benefit:Benefitor(for,$var1) ^2_Benefit:Act(for,$var0)
```

we obtain

```
^2_Benefit:Benefitor(for,Mafia)
^2_Benefit:Act(for,killing)

^2_Benefit:Benefitor(for_1,government)
^2_Benefit:Act(for_1,killing_1)
```

Here the first argument of the output relationships allows us to correctly associate the different acts of killing with the different benefitors

44.5.5 Comparatives and Phantom Nodes

Next, a bit of subtlety is needed to deal with sentences like

Mike eats more cookies than Ben.

which RelEx handles via

```
_subj(eat, Mike)
_obj(eat, cookie)
more(cookie, $cVar0)
$cVar0(Ben)
```

Then a RelEx2FrameNet mapping rule such as:

```
IF
_subj(eat,$var0)
_obj(eat,$var1)
more($var1,$cVar0)
$cVar0($var2)
THEN
^2_AsymmetricEvaluativeComparison:ProfiledItem(more, $var1)
^2_AsymmetricEvaluativeComparison:StandardItem(more, $var1_1)
^2_AsymmetricEvaluativeComparison:Valence(more, more)
^1_Ingestion:Ingestor(eat,$var0)
^1_Ingestion:Ingested(eat,$var1)
^1_Ingestion:Ingestor(eat_1,$var2)
^1_Ingestion:Ingested(eat_1,$var1_1)
```

applies, which embodies the commonsense intuition about comparisons regarding eating. (Note that we have introduced a new frame AsymmetricEvaluativeComparison here, by analogy to the standard FrameNet frame Evaluative_comparison.)

Note also that the above rule may be too specialized, though it's not incorrect. One could also try more general rules like

```
IF
%Agent($var0)
%Agent($var1)
_subj($var3,$var0)
_obj($var3,$var1)
more($var1,$cVar0)
$cVar0($var2)
THEN
^2_AsymmetricEvaluativeComparison:ProfiledItem(more, $var1)
^2_AsymmetricEvaluativeComparison:StandardItem(more, $var1_1)
^2_AsymmetricEvaluativeComparison:Valence(more, more)
_subj($var3,$var0)
_obj($var3,$var1)
_subj($var3_1,$var2)
_obj($var3_1,$var1_1)
```

However, this rule is a little different than most RelEx2Frame rules, in that it produces output that then needs to be processed by the RelEx2Frame rulebase a second time. There's nothing wrong with this, it's just an added layer of complexity.

44.6 Frame2Atom

The next step in the current OpenCog NLP comprehension pipeline is to translate the output of RelEx2Frame into Atoms. This may be done in a variety of ways; the current Frame2Atom script embodies one approach that has proved workable, but is certainly not the only useful one.

The Node types currently used in Frame2Atom are:

- WordNode
- ConceptNode
  - DefinedFrameNode
  - DefinedLinguisticConceptNode
- PredicateNode
  - DefinedFrameElementNode
  - DefinedLinguisticRelationshipNode
- SpecificEntityNode

The special node types

- DefinedFrameNode
- DefinedFrameElementNode

have been created to correspond to FrameNet frames and elements respectively (or frames and elements drawn from similar resources to FrameNet, such as our own frame dictionary).

Similarly, the special node types

- DefinedLinguisticConceptNode
- DefinedLinguisticRelationshipNode

have been created to correspond to RelEx unary and binary relationships respectively.

The "defined" is in the names because once we have a more advanced CogPrime system, it will be able to learn its own frames, frame elements, linguistic concepts and relationships. But what distinguishes these "defined" Atoms is that they have names which correspond to specific external resources.

The Link types we need for Frame2Atom are:

- InheritanceLink
- ReferenceLink (current using WRLink aka "word reference link")
- FrameElementLink

ReferenceLink is a special link type for connecting concepts to the words that they refer to. (This could be eliminated via using more complex constructs, but it's a very common case so for practical purposes it makes sense to define it as a link type.)

FrameElementLink is a special link type connecting a frame to its element. Its semantics (and how it could be eliminated at cost of increased memory and complexity) will be explained below.

44.6.1 Examples of Frame2Atom

Below follow some examples to illustrate the nature of the mapping intended. The examples include a lot of explanatory discussion as well.

Note that, in these examples, $[n]$ denotes an Atom with AtomHandle n. All Atoms have Handles, but Handles are only denoted in cases where this seems useful. (In the XML representation used in the current OpenCogPrime impelmentation, these are replaced by UUID's)

The notation

$$WordNode\#pig$$

denotes a WordNode with name pig, and a similar convention is used for other AtomTypes whose names are useful to know.

These examples pertain to fragments of the parse

```
Ben slowly ate the fat chickens.

A:_advmod:V(slowly:A, eat:V)
N:_nn:N(fat:N, chicken:N)
N:definite(Ben:N)
N:definite(chicken:N)
N:masculine(Ben:N)
N:person(Ben:N)
N:plural(chicken:N)
N:singular(Ben:N)
V:_obj:N(eat:V, chicken:N)
V:_subj:N(eat:V, Ben:N)
V:past(eat:V)

^1_Ingestion:Ingestor(eat,Ben)
^1_Temporal_colocation:Event(past,eat)
^1_Ingestion:Ingestibles(eat,chicken)
^1_Activity:Agent(subject,Ben)
^1_Activity:Activity(verb,eat)
^1_Transitive_action:Event(verb,eat)
^1_Transitive_action:Patient(object,chicken)
```

44.6.1.1 Example 1

$$\_obj(eat, chicken)$$

would map into

```
EvaluationLink
   DefinedLinguisticRelationshipNode #_obj
   ListLink
        ConceptNode [2]
        ConceptNode [3]

InheritanceLink
   [2]
   ConceptNode [4]

InheritanceLink
   [3]
   ConceptNode [5]

ReferenceLink [6]
   WordNode #eat  [8]
   [4]

ReferenceLink [7]
   WordNode #chicken [9]
   [5]
```

Please note that the Atoms labeled 4,5,6,7,8,9 would not normally have to be created when entering the relationship

$$\_obj(eat, chicken)$$

into the AtomTable. They should already be there, assuming the system already knows about the concepts of eating and chickens. These would need to be newly created only if the system had never seen these words before.

For instance, the Atom [2] represents the specific instance of "eat" involved in the relationship being entered into the system. The Atom [4] represents the general concept of "eat", which is what is linked to the word "eat."

Note that a very simple step of inference, from these Atoms, would lead to the conclusion

```
EvaluationLink
   DefinedLinguisticRelationshipNode #_obj
   ListLink
        ConceptNode [4]
        ConceptNode [5]
```

which represents the general statement that chickens are eaten. This is such an obvious and important step, that perhaps as soon as the relationship $\_obj(eat, chicken)$ is entered into the system, it should immediately be carried out (i.e. that link if not present should be created, and if present should have its truth value updated). This is a choice to be implemented in

the specific scripts or schema that deal with ingestion of natural language text.

44.6.1.2 Example 2

$$masculine(Ben)$$

would map into

```
InheritanceLink
    SpecificEntityNode [40]
    DefinedLinguisticConceptNode #masculine

InheritanceLink
   [40]
   [10]

ReferenceLink
   WordNode #Ben
   [10]
```

44.6.1.3 Example 3

The mapping of the RelExToFrame output

$$Ingestion : Ingestor(eat, Ben)$$

would use the existing Atoms

```
DefinedFrameNode #Ingestion [11]
DefinedFrameElementNode #Ingestion:Ingestor [12]
```

which would be related via

```
FrameElementLink [11] [12]
```

(Note that FrameElementLink may in principle be reduced to more elementary PLN link types.)

Note that each FrameNet frame contains some core elements and some optional elements. This may be handled by giving core elements links such as

```
FrameElementLink F E <1>
```

and optional ones links such as

```
FrameElementLink F E <.7>
```

Getting back to the example at hand, we would then have

```
InheritanceLink [2] [11]
```

(recall, [2] is the instance of eating involved in Example ; and, [11] is the Ingestion frame), which says that this instance of eating is an instance of ingestion. (In principle, some instances of eating might not be instances of ingestion ... or more generally, we can't assume that all instances of a given concept will always associate with the same FrameNodes.... This could be assumed only if we assumed all word-associated concepts were disambiguated to a single known FrameNet frame, but this can't be assumed, especially if later on we want to use cognitive processes to do sense disambiguation....)

We would then also have links denoting the role of Ben as an Ingestor in the frame-instance [2], i.e.

```
EvaluationLink
   DefinedFrameElementNode #Ingestion:Ingestor [12]
   ListLink
        [2]
        [40]
```

This says that the specific instance of Ben observed in that sentence ([4]) served the role of Ingestion:Ingestor in regard to the frame-instance [2] (which is an instance of eating, which is known to be an instance of the frame of Ingestion).

44.6.2 Issues Involving Disambiguation

Right now, OpenCogPrimes RelEx2Frame rulebase is far from adequately large (there are currently around 5000 rules) and the link parser and RelEx are also imperfect. The current OpenCog NLP system does work, but for complex sentences it tends to generate too many interpretations of each sentence – "parse selection" or more generally "interpretation selection" is not yet adequately addressed. This is a tricky issue that can be addressed to some extent via statistical linguistics methods, but we believe that to solve it convincingly and thoroughly will require more cognitively sophisticated methods.

The most straightforward way to approach it statistically is to process a large number of sentences, and then tabulate co-occurrence probabilities of different relationships across all the sentences. This allows one to calculate the probability of a given interpretation conditional on the corpus, via looking at the probabilities of the combinations of relationships in the interpretation. This may be done using a Bayes Net or using PLN – in any case the problem is one of calculating the probability of a conjunction of terms based on knowledge regarding the probabilities of various sub-conjunctions. As this method doesn't require marked-up training data, but is rather purely

unsupervised, it's feasible to apply it to a very large corpus of text – the only cost is computer time.

What the statistical approach won't handle, though, are the more conceptually original linguistic constructs, containing combinations that didn't occur frequently in the system's training corpus. It will rate innovative semantic constructs as unlikely, which will lead it to errors sometimes – errors of choosing an interpretation that seems odd in terms of the sentence's real-world interpretation, but matches will with things the system has seen before. The only way to solve this is with genuine understanding – with the system reasoning on each of the interpretations and seeing which one makes more sense. And this kind of reasoning generally requires some relevant commonsense background knowledge – which must be gained via experience, reading and conversing, or from a hand-coded knowledge base, or via some combination of the above.

Related issues also involving disambiguation include word sense disambiguation (words with multiple meanings) and anaphor resolution (recognizing the referents of pronouns, and of nouns that refer to other nouns, etc.).

The current RelEx system contains a simple statistical parse ranker (which rates a parse higher if the links it includes occur more frequently in a large parsed corpus), statistical methods for word sense disambiguation [Mih07] inspired by those in Rada Mihalcea's work [SM09], and an anaphor resolution algorithm based on the classic Hobbs Algorithm (customized to work with the link parser) [Hob78]. While reasonably effective in many cases, from an AGI perspective these must all be considered "stopgaps" to be replaced with code that handles these tasks using probabilistic inference. It is conceptually straightforward to replace statistical linguistic algorithms with comparable PLN-based methods, however significant attention must be paid to code optimization as using a more general algorithm is rarely as efficient as using a specialized one. But once one is handling things in PLN and the Atomspace rather than in specialized computational linguistics code, there is the opportunity to use a variety of inference rules for generalization, analogy and so forth, which enables a radically more robust form of linguistic intelligence.

44.7 Link2Atom: A Semi-Supervised Alternative to RelEx and RelEx2Frame

This section describes an alternative approach to the RelEx / RelEx2Frame approach described above, which is in the midst of implementation at time of writing. This alternative represents a sort of midway point between the rule-based RelEx / RelEx2Frame approach, and a conceptually ideal fully experiential learning based approach.

The motivations underlying this alternative approach have been to create an OpenCog NLP system with the capability

- to support simple dialogue in a video game like world, and a robot system
- to leverage primarily semi-supervised experiential learning
- to replace the RelEx2Frame rules, which are currently problematic, with a different way of mapping syntactic relationships into Atoms, that is still reasoning and learning friendly
- to require only relatively modest effort for implementation (not multiple human-years)

The latter requirement ruled out a pure "learn language from experience with no aid from computational linguistics tools" approach, which may well happen within OpenCog at some point.

44.8 Mapping Link Parses into Atom Structures

The core idea of the new approach is to learn "Link2Atom" rules that map *link parses* into *Atom structures*. These rules may then be automatically reversed to form Atom2Link rules, which may be used in language generation.

Note that this is different from the RelEx approach as currently pursued (the "old approach"), which contains

- one set of rules (the RelEx rules) mapping link parses into semantic relation-sets ("RelEx relation-sets" or rel-sets)
- another set of rules (the RelEx2Frame rules) mapping rel-sets into FrameNet-based relation-sets
- another set of rules (the Frame2Atom rules) mapping FrameNet-based relation-sets into Atom-sets

In the old approach, all the rules were hand-coded. In the new approach

- nothing *needs* to be hand-coded (except the existing link parser dictionary); the rules can be learned from a corpus of (link-parse, Atom-set) pairs. This corpus may be human-created; or may be derived via a system's experience in some domain where sentences are heard or read, and can be correlated with observed nonlinguistic structures that can be described by Atoms.
- in practice, some hand-coded rules are being created to map RelEx rel-sets into Atom-sets directly (bypassing RelEx2Frame) in a simple way. These rules will be used, together with RelEx, to create a large corpus of (link parse, Atom-set) pairs, which will be used as a training corpus. This training corpus will have more errors than a hand-created corpus, but will have the compensating advantage of being significantly larger than any hand-created corpus would feasibly be.

In the old approach, NL generation was done by using a pattern-matching approach, applied to a corpus of (link parse, rel-set) pairs, to mine rules

mapping rel-sets to sets of link parser links. This worked to an extent, but the process of piecing together the generated sets of link parser links to form coherent "sentence parses" (that could then be turned into sentences) turned out to be subtler than expected, and appeared to require an escalatingly complex set of hand-coded rules, to be extended beyond simple cases.

In the new approach, NL generation is done by explicitly reversing the mapping rules learned for mapping link parses into Atom sets. This is possible because the rules are explicitly given in a form enabling easy reversal; whereas in the old approach, RelEx transformed link parses into rel-sets using a process of successively applying many rules to an ornamented tree, each rule acting on variables ("ornaments") deposited by previous rules. Put simply, RelEx transformed link parses into rel-sets via imperative programming, whereas in the new approach, link parses are transformed into Atom-sets using learned rules that are *logical* in nature. The movement from imperative to logical style dramatically eases automated rule reversal.

44.8.1 Example Training Pair

For concreteness, an example (link parse, Atom-set) pair would be as follows. For the sentence "Trains move quickly", the link parse looks like

```
Sp(trains, move)
MVa(move, quickly)
```

whereas the Atom-set looks like

```
Inheritance
   move_1
   move

Evaluation
   move_1
   train

 Inheritance
   move_1
   quick
```

Rule learning proceeds, in the new approach, from a corpus consisting of such pairs.

44.9 Making a Training Corpus

44.9.1 Leveraging RelEx to Create a Training Corpus

To create a substantial training corpus for the new approach, we are leveraging the existence of RelEx. We have a large corpus of sentences parsed by the link parser and then processed by RelEx. A new collection of rules is being created, RelEx2Atom, that directly translates RelEx parses into Atoms, in a simple way, embodying the minimal necessary degree of disambiguation (in a sense to be described just below). Using these RelEx2Atom rules, one can transform a corpus of (link parse, RelEx rel-set) triples into a corpus of (link parse, Atom-set) pairs – which can then be used as training data for learning Link2Atom rules.

44.9.2 Making an Experience Based Training Corpus

An alternate approach to making a training corpus would be to utilize a virtual world such as the Unity3D world now being used for OpenCog game AI research and development.

A human game-player could create a training corpus by repeated:

- typing in a sentence
- indicating, via the graphic user interface, which entities or events in the virtual world were referred to by the sentence

Since OpenCog possesses code for transforming entities and events in the virtual world into Atom-sets, this would implicitly produce a training corpus of (sentence, Atom-set) pairs, which using the link parser could then be transformed into (link parse, Atom-set) pairs.

44.9.3 Unsupervised, Experience Based Corpus Creation

One could also dispense with the explicit reference-indication GUI, and just have a user type sentences to the AI agent as the latter proceeds through the virtual world. The AI agent would then have to figure out what specifically the sentences were referring to – maybe the human-controlled avatar is pointing at something; maybe one thing recently changed in the game world and nothing else did; etc. This mode of corpus creation would be reasonably similar to human first language learning in format (though of course there are many differences from human first language learning in the overall approach, for instance we are assuming the link parser, whereas a human language learner

has to learn grammar for themselves, based on complex and ill-understood genetically encoded prior probabilistic knowledge regarding the likely aspects of the grammar to be learned).

This seems a very interesting direction to explore later on, but at time of writing we are proceeding with the RelEx-based training corpus, for sake of simplicity and speed of development.

44.10 Limiting the Degree of Disambiguation Attempted

The old approach is in a sense more ambitious than the new approach, because the RelEx2Frame rules attempt to perform a deeper and more thorough level of semantic disambiguation than the new rules. However, the RelEx2Frame rule-set in its current state is too "noisy" to be really useful; it would need dramatic improvement to be helpful in practice. The key difference is that,

- In the new approach, the syntax-to-semantics mapping rules attempt *only* the disambiguation that needs to be done to get the structure of the resultant Atom-set correct. Any further disambiguation is left to be done later, by MindAgents acting on the Atom-sets after they've already been placed in the AtomSpace.
- In the old approach, the RelEx2Frame rules attempted, in many cases, to disambiguate between different meanings beyond the level needed to disambiguate the structure of the Atom-set

To illustrate the difference, consider the sentences

- Love moves quickly
- Trains move quickly

These sentences involve different senses of "move" – change in physical location, versus a more general notion of progress. However, both sentences map to the same basic conceptual structure, e.g.

```
Inheritance
   move_1
   move

Evaluation
   move_1
   train

 Inheritance
   move_1
   quick
```

versus

```
Inheritance
   move_2
   move

Evaluation
   move_2
   love

 Inheritance
   move_2
   quick
```

The RelEx2Frame rules try to distinguish between these cases via, in effect, associating the two instances move_1 and move_2 with different frames, using hand-coded rules that map RelEx rel-sets into appropriate Atom-sets defined in terms of FrameNet relations. This is not a useless thing to do; however, doing it well requires a very large and well-honed rule-base. Cyc's natural language engine attempts to do something similar, though using a different parser than the link parser and a different ontology than FrameNet; it does a much better job than the current version of RelEx2Frame, but still does a surprisingly incomplete job given the massive amount of effort put into sculpting the relevant rule-sets.

The new approach does not try to perform this kind of disambiguation prior to mapping things into Atom-sets. Rather, this kind of disambiguation is left for inference to do, after the relevant Atoms have already been placed in the AtomSpace. The rule of thumb is: Do precisely the disambiguation needed to map the parse into a compact, simple Atom-set, whose component nodes correspond to English words. Let the disambiguation of the meaning of the English words be done by some other process acting on the AtomSpace.

44.11 Rule Format

To represent Link2Atom rules, it is convenient to represent link parses as Atom-sets. Each element of the training corpus will then be of the form (Atom set representing link parse, Atom-set representing semantic interpretation). Link2Atom rules are then rules mapping Atom-sets to Atom-sets.

Broadly speaking, the format of a Link2Atom rule is then

```
Implication
    Atom-set representing portion of link parse
    Atom-set representing portion of semantic interpretation
```

44.11.1 Example Rule

A simple example rule would be

```
Implication
    Evaluation
         Predicate: Sp
         \$V1
         \$V2
    Evaluation
         \$V2
         \$V1
```

This rule, in essence, maps verbs into predicates that take their subjects as arguments.

On the other hand, an Atom2Link rule would look like the reverse:

```
Implication
    Atom-set representing portion of link parse
    Atom-set representing portion of semantic interpretation
```

Our currentapproach is to begin with Link2Atom rules, because, due to the nature of natural language, these rules will tend to be more certain. That is: it is more strongly the case in natural languages that each syntactic construct maps into a small set of semantic structures, than that each semantic structure is realizable only via a small set of syntactic constructs. There are usually more ways structurally different, reasonably sensible ways to say an arbitrary thought, than there are structurally different, reasonably sensible ways to interpret an arbitrary sentence. Because of this fact about language, the design of the Atom-sets in the corpus is based on the principle of finding an Atom structure that most simply represents the meaning of the sentence corresponding to each given link parse. Thus, there will be many Link2Atom rules with a high degree of certitude attached to them. On the other hand, the Atom2Link rules will tend to have less certitude, because there may be many different syntactic ways to realize a given semantic expression.

44.12 Rule Learning

Learning of Link2Atom rules may be done via any algorithm that is able to search rule space for rules of the proper format with high truth value as evaluated across the training set. Currently we are experimenting with using OpenCogPrimes frequent subgraph mining algorithm in this context. MOSES could also potentially be used to learn Link2Atom rules. One suspects that MOSES might be better than frequent subgraph mining for learning complex

rules, but based on preliminary experimentation, frequent subgraph mining seems fine for learning the simple rules involved in simple sentences.

PLN inference may also be used to generate new rules by combining previous ones, and to generalize rules into more abstract forms.

44.13 Creating a Cyc-Like Database via Text Mining

The discussion of these NL comprehension mechanisms leads naturally to one interesting potential application of the OpenCog NL comprehension pipeline – which is only indirectly related to CogPrime , but would create a valuable resource for use by CogPrime if implemented. The possibility exists to use the OpenCog NL comprehension system to create a vaguely *Cyc-like database of common-sense rules*.

The approach would be as follows:

1. Get a corpus of text
2. Parse the text using OpenCog (RelEx or Link2Atom)
3. Mine logical relationships among Atomrelationships from the data thus produced, using greedy data-mining, MOSES, or other methods

These mined logical relationships will then be loosely analogous to the rules the Cyc team have programmed in. For instance, there will be many rules like:

```
# IF _subj(understand,$var0) THEN ^1_Grasp:Cognizer(understand,$var0)
# IF _subj(know,$var0) THEN ^1_Grasp:Cognizer(understand,$var0)
```

So statistical mining would learn rules like

```
IF ^1_Mental_property(stupid) & ^1_Mental_property:Protagonist($var0)
  THEN ^1_Grasp:Cognizer(understand,$var0) <.3>
IF ^1_Mental_property(smart) & ^1_Mental_property:Protagonist($var0)
  THEN ^1_Grasp:Cognizer(understand,$var0) <.8>
```

which means that stupid people mentally grasp less than smart people do.

Note that these commonsense rules would come out automatically probabilistically quantified.

Note also that to make such rules come out well, one needs to do some (probabilistic) synonym-matching on nouns, adverbs and adjectives, e.g. so that mentions of "smart", "intelligent", "clever", etc. will count as instances of

```
^1_Mental_property(smart)
```

By combining probabilistic synonym matching on words, with mapping RelEx output into FrameNet input, and doing statistical mining, it should be possible to build a database like Cyc but far more complete and with coherent probabilistic weightings.

Although this way of building a commonsense knowledge base requires a lot of human engineering, it requires far less than something like Cyc. One "just" needs to build the RelEx2FrameNet mapping rules, not all the commonsense knowledge relationships directly – those come from text. We do not advocate this as a solution to the AGI problem, but merely suggest that it could produce a large amount of useful knowledge to feed into an AGI's brain.

And of course, the better an AI one has, the better one can do the step labeled "Rank the parses and FrameNet interpretations using inference or heuristics or both." So there is a potential virtuous cycle here: more commonsense knowledge mined helps create a better AI mind, which helps mine better commonsense knowledge, etc.

44.14 PROWL Grammar

We have described the crux of the NL comprehension pipeline that is currently in place in the OpenCog codebase, plus some ideas for fairly moderate modifications or extensions. This section is a little more speculative, and describes an alternative approach that fits better with the overall CogPrime design, which however has not yet been implemented. The ideas given here lead more naturally to a design for experience-based language learning and processing, a connection that will be pointed out in a later section.

What we describe here is a partially-new theory of language formed via combining ideas from three sources: Hudson's Word Grammar [Hud90, Hud07], Sleator and Temperley's link grammar, and Probabilistic Logic Networks. Reflecting its origin in these three sources, we have named the new theory PROWL grammar, meaning PRObabilistic Word Link Grammar. We believe PROWL has value purely as a conceptual approach to understanding language; however, it has been developed largely from the standpoint of computational linguistics – as part of an attempt to create a framework for computational language understanding and generation that both

1. yields broadly adequate behavior based on hand-coding of "expert rules" such as grammatical rules, combined with statistical corpus analysis
2. integrates naturally with a broader AI framework that combines language with embodied social, experiential learning, that ultimately will allow linguistic rules derived via expert encoding and statistical corpus analysis to be replaced with comparable, more refined rules resulting from the system's own experience

PROWL has been developed as part of the larger CogPrime project; but, it is described in this section mostly in a CogPrime -independent way, and is intended to be independently evaluable (and, hopefully, valuable).

As an integration of three existing frameworks, PROWL could be presented in various different ways. One could choose any one of the three components as an initial foundation, and then present the combined theory as an expansion/modification of this component. Here we choose to present it as an expansion/modification of Word Grammar, as this is the way it originated, and it is also the most natural approach for readers with a linguistics background. From this perspective, to simplify a fair bit, one may describe PROWL as consisting of Word Grammar with three major changes:

1. Word Grammar's network knowledge representation is replaced with a richer PLN-based network knowledge representation.
    a. This includes, for instance, the replacement of Word Grammar's single "isa" relationship type with a more nuanced collection of logically distinct probabilistic inheritance relationship types
2. Going along with the above, Word Grammar's "default inheritance" mechanism is replaced by an appropriate PLN control mechanism that guides the use of standard PLN inference rules
    a. This allows the same default-inheritance based inferences that Word Grammar relies upon, but embeds these inferences in a richer probabilistic framework that allows them to be integrated with a wide variety of other inferences
3. Word Grammar's small set of syntactic link types is replaced with a richer set of syntactic link types as used in Link Grammar
    a. The precise optimal set of link types is not clear; it may be that the link grammar's syntactic link type vocabulary is larger than necessary, but we also find it clear that the current version of Word Grammar's syntactic link type vocabulary is smaller than feasible (at least, without the addition of large, new, and as yet unspecified ideas to Word Grammar)

In the following subsections we will review these changes in a little more detail. Basic familiarity with Word Grammar, Link Grammar and PLN is assumed.

Note that in this section we will focus mainly on those issues that are somehow nonobvious. This means that a host of very important topics that come along with the Word Grammar / PLN integration are not even mentioned. The way Word Grammar deals with morphology, semantics and pragmatics, for instance, seems to us quite sensible and workable – and doesn't really change at all when you integrate Word Grammar with PLN, except that Word Grammar's crisp isa links become PLN-style probabilistic Inheritance links.

44.14.1 Brief Review of Word Grammar

Word Grammar is a theory of language structure which Richard Hudson began developing in the early 1980's [**?**]. While partly descended from Systemic Functional Grammar, there are also significant differences. The main ideas of Word Grammar are as follows †:

- It presents language as a network of knowledge, linking concepts about words, their meanings, etc. - e.g. the word "dog" is linked to the meaning 'dog', to the form /dog/, to the word-class 'noun', etc.
- If language is a network, then it is possible to decide what kind of network it is (e.g. it seems to be a scale-free small-world network)
- It is monostratal - only one structure per sentence, no transformations.
- It uses word-word dependencies - e.g. a noun is the subject of a verb.
- It does not use phrase structure - e.g. it does not recognise a noun phrase as the subject of a clause, though these phrases are implicit in the dependency structure.
- It shows grammatical relations/functions by explicit labels - e.g. 'subject' and 'object'.
- It uses features only for inflectional contrasts that are mentioned in agreement rules - e.g. number but not trense or transitivity.
- It uses default inheritance, as a very general way of capturing the contrast between 'basic' or 'underlying' patterns and 'exceptions' or 'transformations' - e.g. by default, English words follow the word they depend on, but exceptionally subjects precede it; particular cases 'inherit' the default pattern unless it is explicitly overridden by a contradictory rule.
- It views concepts as prototypes rather than 'classical' categories that can be defined by necessary and sufficient conditions. All characteristics (i.e. all links in the network) have equal status, though some may for pragmatic reasons be harder to override than others.
- In this network there are no clear boundaries between different areas of knowledge - e.g. between 'lexicon' and 'grammar', or between 'linguistic meaning' and 'encyclopedic knowledge'; language is not a separate module of cognition.
- In particular, there is no clear boundary between 'internal' and 'external' facts about words, so a grammar should be able to incorporate sociolinguistic facts - e.g. the speaker of "sidewalk" is an American.

† the following list is paraphrased with edits from http://www.phon.ucl.ac.uk/home/dick/wg.htm downloaded on June 27 2010

44.14.2 Word Grammar's Logical Network Model

Word Grammar presents an elegant framework in which all the different aspects of language are encompassed within a single knowledge network. Representationally, this network combines two key aspects:

1. Inheritance (called is-a) is explicitly represented
2. General relationships between n-ary predicates and their arguments, including syntactic relationships, are explicitly represented

Dynamically, the network contains two key aspects:

1. An inference rule called "default inheritance"
2. Activation-spreading, similar to in a neural network or standard semantic network

The similarity between Word Grammar and CogPrime is fairly strong. In the latter, inheritance and generic predicate-argument relationships are explicitly represented; and, a close analogue of activation spreading is present in the "attention allocation" subsystem. As in Word Grammar, important cognitive phenomena are grounded in the symbiotic combination of logical-inference and activation-spreading dynamics.

At the most general level, the reaction of the Word Grammar network to any situation is proposed to involve three stages:

1. Node creation and identification: of nodes representing the situation as understood, in its most relevant aspects
2. Where choices need to be made (e.g. where an identified predicate needs to choose which other nodes to bind to as arguments), activation spreading is used, and the most active eligible argument is utilized (this is called "best fit binding")
3. Default inheritance is used to supply new links to the relevant nodes as necessary

Default inheritance is a process that relies on the placement of each node in a hierarchy (dag) of isa links. The basic idea is as follows. Suppose one has a node N, and a predicate f(N,L), where L is another argument or list of arguments. Then, if the truth value of f(N,L) is not explicitly stored in the network, N inherits the value from any ancestor A in the dag so that: f(A,L) is explicitly stored in the network; and there is not any node P inbetween N and A for which f(P,L) is explicitly stored in the network. Note that multiple inheritance is explicitly supported, and in cases where this leads to multiple assignments of truth values to a predicate, confusion in the linguistic mind may ensue. In many cases the option coming from the ancestor with the highest level of activity may be selected.

Our suggestion is that Word Grammar's network representation may be replaced with PLN's logical network representation without any loss, and

with significant gain. Word Grammar's network representation has not been fleshed out as thoroughly as that of PLN, it does not handle uncertainty, and it is not associated with general mechanisms for inference. The one nontrivial issue that must be addressed in porting Word Grammar to the PLN representation is the role of default inheritance in Word Grammar. This is covered in the following subsection.

The integration of activation spreading and default inheritance proposed in Word Grammar, should be easily achievable within CogPrime assuming a functional attention allocation subsystem.

44.14.3 Link Grammar Parsing vs Word Grammar Parsing

From a CogPrime /PLN point of view, perhaps the most striking original contribution of Word Grammar is in the area of syntax parsing. Word Grammar's treatment of morphology and semantics is, basically, exactly what one would expect from representing such things in a richly structured semantic network. PLN adds much additional richness to Word Grammar via allowing nuanced representation of uncertainty, which is critical on every level of the linguistic hierarchy – but this doesn't change the fundamental linguistic approach of Word Grammar. Regarding syntax processing, however, Word Grammar makes some quite specific and unique hypotheses, which if correct are very valuable contributions.

The conceptual assumption we make here is that syntax processing, while carried out using generic cognitive processes for uncertain inference and activation spreading, also involves some highly specific constraints on these processes. The extent to which these constraints are learned versus inherited is yet unknown, and for the subtleties of this issue the reader is referred to [EBJ+97]. Word Grammar and Link Grammar are then understood as embodying different hypotheses regarding what these constraints actually are.

It is interesting to consider the contributions of Word Grammar to syntax parsing via comparing it to Link Grammar.

Note that Link Grammar, while a less comprehensive conceptual theory than Word Grammar, has been used to produce a state-of-the-art syntax parser, which has been incorporated into a number of other software systems including OpenCog. So it is clear that the Link Grammar approach has a great deal of pragmatic value. On the other hand, it also seems clear that Link Grammar has certain theoretical shortcomings. It deals with many linguistic phenomena very elegantly, but there are other phenomena for which its approach can only be described as "hacky."

Word Grammar contains fewer hacks than Link Grammar, but has not yet been put to the test of large-scale computational implementation, so it's not yet clear how many hacks would need to be added to give it the relatively

broad coverage that Link Grammar currently has. Our own impression is that to make Word Grammar actually work as the foundation for a broad-coverage grammar parser (whether standalone, or integrated into a broader artificial cognition framework), one would need to move it somewhat in the direction of link grammar, via adding a greater number of specialized syntactic link types (more on this shortly). There are in fact concrete indications of this in [Hud07].

The Link Grammar framework may be decomposed into three aspects:

1. The link grammar dictionary, which for each word in English, contains a number of links of different types. Some links point left, some point right, and each link is labeled. Furthermore, some links are required and others are optional.
2. The "no-links-cross" constraint, which states that the correct parse of a sentence will involve drawing links between words, in such a way that all the required links of each word are fulfilled, and no two links cross when the links are depicted in two dimensions
3. A processing algorithm, which involves first searching the space of all possible linkages among the words in a sentence to find all complete linkages that obey the no-links-cross constraint; and then applying various postprocessing rules to handle cases (such as conjunctions) that aren't handled properly by this algorithm

In PROWL, what we suggest is that

1. The link grammar dictionary is highly valuable and provides a level of linguistic detail that is not present in Word Grammar; and, we suggest that in order to turn Word Grammar into a computationally tractable system, one will need something at least halfway between the currently minimal collection of syntactic link types used in Word Grammar and the much richer collection used in Link Grammar
2. The no-links-cross constraint is an approximation of a deeper syntactic constraint ("landmark transitivity") that has been articulated in the most recent formulations of Word Grammar. Specifically: when a no-links-crossing parse is found, it is correct according to Word Grammar; but Word Grammar correctly recognizes some parses that violate this constraint
3. The Link Grammar parsing algorithm is not cognitively natural, but is effective in a standalone-parsing framework. The Word Grammar approach to parsing is cognitively natural, but as formulated could only be computationally implemented in the context of an already-very-powerful general intelligence system. Fortunately, various intermediary approaches to parsing seem possible.

44.14.3.1 Using Landmark Transitivity with the Link Grammar Dictionary

An earlier version of Word Grammar utilized a constraint called "no tangled links" which is equivalent to the link parser's "no links cross" constraint. In the new version of Word Grammar this is replaced with a subtler and more permissive constraint called "landmark transitivity." While in Word Grammar, landmark transitivity is used with a small set of syntactic link types, there is no reason why it can't be used with the richer set of link types that Link Grammar provides. In fact, this seems to us a probably effective method of eliminating most or all of the "postprocessing rules" that exist in the link parser, and that constitute the least elegant aspect of the Link Grammar framework.

The first foundational concept, on the path to the notion of landmark transitivity, is the notion of a syntactic parent. In Word Grammar each syntactic link has a parent end and a child end. In a dependency grammar context, the notion is that the child depends upon the parent. For instance, in Word Grammar, in the link between a noun and an adjective, the noun is the parent.

To apply landmark transitivity in the context of the Link Grammar, one needs to provide some additional information regarding each link in the Link Grammar dictionary. One needs to specify which end of each of the link grammar links is the "parent" and which is the "child." Examples of this kind of markup are as follows (with P shown by the parent):

```
                    P

S: subject-noun ------- finite verb


P

O: transitive verb ----- direct or indirect object


            P

D: determiner ----- noun



         P

MV: verb ----- verb modifier
```

```
          P
J: preposition ----- object

     P
ON: on ----- time-expression

      P
M: noun ----- modifiers
```

In some cases a word may have more than one parent. In this case, the rule is that the landmark is the one that is superordinate to all the other parents. In the rare case that two words are each others' parents, then either may serve as the landmark.

The concept of a parent leads naturally into that of a landmark. The first rule regarding landmarks is that a parent is a landmark for its child. Next, two kinds of landmarks are introduced: Before landmarks (in which the child is before the parent) and After landmarks (in which the child is after the parent). The Before/After distinction should be obvious in the Link Grammar examples given above.

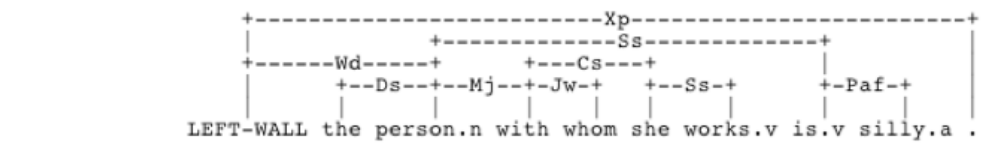
The landmark transitivity rule, then, has two parts. If A is a landmark for B, of subtype L (where L is either Before or After), then

1. **Subordinate transitivity** says that if B is a landmark for C, then A is also a type-L landmark for C
2. **Sister transitivity** says that if A is a landmark for C, then B is also a landmark for C

Finally, there are some special link types that cause a word to depend on its grandparents or higher ancestors as well as its parents. I note that these are not treated thoroughly in (Hudson, 2007); one needs to look to the earlier, longer and rarer work [Hud90]. Some questions are dealt with this way. Another example is what in Word Grammar is called a "proxy link", as occurs between "with" and "whom" in

The person with whom she works

The link parser deals with this particular example via a Jw link



so to apply landmark transitivity in the context of the Link Grammar, in this case, it seems one would need to implement the rule that in the case of two words connected by a Jw-link, the child of one of the words is also the child of the other. Handling other special cases like this in the context of Link Grammar seems conceptually unproblematic, though naturally some hidden rocks may appear. Basically a list needs to be made of which kinds of link parser links embody proxy relationships for which other kinds of link parser links.

According to the landmark transitivity approach, then, the criterion for syntactic correctness of a parse is that, if one takes the links in the parse and applies the landmark transitivity rule (along with the other special-case "raising" rules we've discussed), one does not arrive at any contradictions (i.e. no situations where A is a Before landmark of B, and

The main problem with the landmark-transitivity constraint seems to be computational tractability. The problem exists for both comprehension and generation, but we'll focus on comprehension here.

To find all possible parses of a sentence using Hudson's landmark-transitivity-based approach, one needs to find all linkages that don't lead to contradictions when used as premises for reasoning based on the landmark-transitivity axioms. This appears to be extremely computationally intensive! So, it seems that Word Grammar style parsing is only computationally feasible for a system that has extremely strong semantic understanding, so as to be able to filter out the vast majority of possible parses on semantic rather than purely syntactic grounds.

On the other hand, it seems possible to apply landmark-transitivity together with no-links-cross, to provide parsing that is both efficient and general. If applying the no-links-cross constraint finds a parse in which no links cross, without using postprocessing rules, then this will always be a legal parse according to the landmark-transitivity rule.

However, landmark-transitivity also allows a lot of other parses that link grammar either needs postprocessing rules to handle, or can't find even with postprocessing rules. So, it would make sense to apply no-links-cross parsing first, but then if this fails, apply landmark-transitivity parsing starting from the partial parses that the former stage produced. This is the approach suggested in PROWL, and a similar approach may be suggested for language generation.

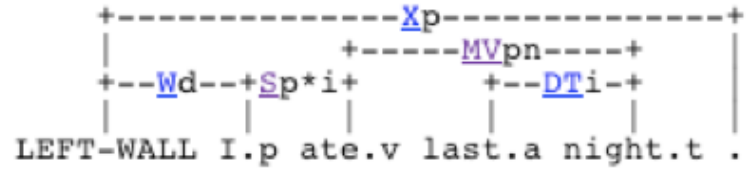
44.14.3.2 Overcoming the Current Limitations of Word Grammar

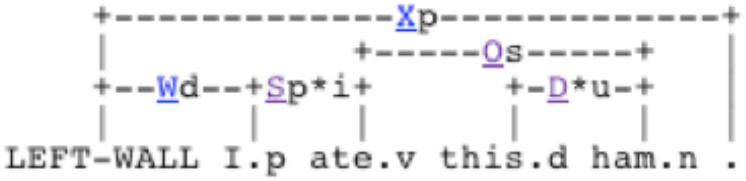
Finally, it is worth noting that expanding the Word Grammar parsing framework to include the link grammar dictionary, will likely allow us to solve some unsolved problems in Word Grammar. For instance, [Hud07] notes that the current formulation of Word Grammar has no way to distinguish the behavior of last vs. this in

I ate last night
I ate this ham

The issue he sees is that in the first case, *night* should be considered the parent of *last*; whereas in the second case, *this* should be considered the parent of *ham*.

The current link parser also fails to handle this issue according to Hudson's intuition:





However, the link grammar framework gives us a clear possibility for allowing the kind of interpretation Hudson wants: just allow *this* to take a left-going O-link, and (in PROWL) let it optionally assume the parent role when involved in a D-link relationship. There are no funky link-crossing or semantic issues here; just a straightforward link-grammar dictionary edit.

This illustrates the syntactic flexibility of the link parsing framework, and also the inelegance – adding new links to the dictionary generally solves syntactic problems, but at the cost of creating more complexity to be dealt with further down the pipeline, when the various link types need to be compressed into a smaller number of semantic relationship types for purposes of actual comprehension (as is done in RelEx, for example). However, as far as we can tell, this seems to be a necessary cost for adequately handling the full complexity of natural language syntax. Word Grammar holds out the hope of possibly avoiding this kind of complexity, but without filling in enough details to allow a clear estimate of whether this hope can ever be fulfilled.

44.14.4 Contextually Guided Greedy Parsing and Generation Using Word Link Grammar

Another difference between Link Grammar and currently utilized, and Word Grammar as described, is the nature of the parsing algorithm. Link Grammar operates in a manner that is fairly traditional among contemporary parsing algorithms: given a sentence, it produces a large set of possible parses, and then it is left to other methods/algorithms to select the right parse, and to form a semantic interpretation of the selected parse. Parse selection may of course involve semantic interpretation: one way to choose the right parse is to choose the one that has the most contextually sensible semantic interpretation. We may call this approach *whole-sentence purely-syntactic parsing*, or WSPS parsing.

One of the nice things about Link Grammar, as compared to many other computational parsing frameworks, is that it produces a relatively small number of parses, compared for instance to typical head-driven phrase-structure grammar parsers. For simple sentences the link parser generally produces only handful of parses. But for complex sentences the link parser can produce hundreds of parses, which can be computationally costly to sift through.

Word Grammar, on the other hand, presents far fewer constraints regarding which words may link to other words. Therefore, to apply parsing in the style of the current link parser, in the context of Word Grammar, would be completely infeasible. The number of possible parses would be tremendous. The idea of Word Grammar is to pare down parses via semantic/pragmatic sensibleness, during the course of the syntax parsing process, rather than breaking things down into two phases (parsing followed by semantic/pragmatic interpretation). Parsing is suggested to proceed forward through a sentence: when a word is encountered, it is linked to the words coming before it in the sentence, in a way that makes sense. If this seems impossible, consistently with the links that have already been drawn in the course of the parsing process, then some backtracking is done and prior choices may be revisited. This approach is more like what humans do when parsing a sentence, and does not have the effect of producing a large number of syntactically possible, semantically/pragmatically absurd parses, and then sorting through them afterwards. It is what we call a *contextually-guided greedy parsing* (CGGP) approach.

For language generation, the link parser and Word Grammar approaches also suggest different strategies. Link Grammar suggests taking a semantic network, then searching holistically for a linear sequence of words that, when link-parsed, would give rise to that semantic network as the interpretation. On the other hand, Word Grammar suggests taking that same semantic network and iterating through it progressively, verbalizing each node of the network as one walks through it, and backtracking if one reaches a point where there

is no way to verbalize the current node consistently with how one has already verbalized the previous nodes.

The main observation we want to make here is that, while Word Grammar by its nature (due to the relative paucity of explicit constraints on which syntactic links may be formed), can operate with CGGP but not WSPS parsing. On the other hand, while Link Grammar is currently utilized with WPSP parsing, there is no reason one can't use it with CGGP parsing just as well. There is no objection to using CGGP parsing together with the link-parser dictionary, nor with the no-links cross constraint rather than the landmark-transitivity constraint (in fact, as noted above, earlier versions of Word Grammar made use of the no-links-cross constraint).

What we propose in PROWL is to use the link grammar dictionary together with the CGGP parsing approach. The WSPS parsing approach may perhaps be useful as a fallback for handling extremely complex and perverted sentences where CGGP takes too long to come to an answer – it corresponds to sentences that are so obscure one has to do really hard, analytical thinking to figure out what they mean.

Regarding constraints on link structure, the suggestion in PROWL is to use the no-links-cross constraint as a first approximation. In comprehension, if no sufficiently high-probability interpretation obeying the no-links-cross constraint is found, then the scope of investigation should expand to include link-structures obeying landmark-transitivity but violating no-links-cross. In generation, things are a little subtler: a list should be kept of link-type combinations that often correctly violate no-links-cross, and when these combinations are encountered in the generation process, then constructs that satisfy landmark-transitivity but not no-links-cross should be considered.

Arguably, the PROWL approach is less elegant than either Link Grammar or Word Grammar considered on its own. However, we are dubious of the proposition that human syntax processing, with all its surface messiness and complexity, is really generated by a simple, unified, mathematically elegant underlying framework. Our goal is not to find a maximally elegant theoretical framework, but rather one that works both as a standalone computational-linguistics system, and as an integrated component of an adaptively-learning AGI system.

44.15 Aspects of Language Learning

Now we finally turn to language learning – a topic that spans the engineered and experiential approaches to NLP. In the experiential approach, learning is required to gain even simple linguistic functionality. In the engineered approach, even if a great deal of linguistic functionality is built in, learning may be used for adding new functionality and modifying the initially given functionality. In this section we will focus on a few aspects of language learn-

ing that would be required even if the current engineered OpenCog comprehension pipeline were completed to a high level of functionality. The more thoroughgoing language learning required for the experiential approach will then be discussed in the following section.

44.15.1 Word Sense Creation

In our examples above, we've frequently referred to ReferenceLinks between WordNodes and ConceptNodes. But, how do these links get built? One aspect of this is the process of *word sense creation*.

Suppose we have a WordNode W that has ReferenceLinks to a number of different ConceptNodes. A common case is that these ConceptNodes fall into clusters, each one denoting a "sense" of the word. The clusters are defined by the following relationships:

1. ConceptNodes within a cluster have high-strength SimilarityLinks to each other
2. ConceptNodes in different clusters have low-strength (i.e. dissimilarity-denoting) SimilarityLinks to each other

When a word is first learned, it will normally be linked only to mutually agreeable ConceptNodes, i.e. there will only be one sense of the word. As more and more instances of the word are seen, however, eventually the WordNode will gather more than one sense. Sometimes different senses are different syntactically, other times they are different only semantically, but are involved in the same syntactic relationships. In the case of a word with multiple senses, most of the relevant feature structure information will be attached to word-sense-representing ConceptNodes, not to WordNodes themselves.

The formation of sense-representing ConceptNodes may be done by the standard clustering and predicate mining processes, which will create such ConceptNodes when there are adequately many Atoms in the system satisfying the criteria represent. It may also be valuable to create a particular SenseMining CIM-Dynamic, which uses the same criteria for node formation as the clustering and predicate mining CIM-Dynamics, but focuses specifically on creating predicates related to WordNodes and their nearby ConceptNodes.

44.15.2 Feature Structure Learning

We've mentioned above the obvious fact that, to intelligently use a feature-structure based grammar, the system needs to be capable of learning new

linguistic feature structures. Probing into this in more detail, we see that there are two distinct but related kinds of feature structure learning:

1. learning the values that features have for particular word senses.
2. learning new features altogether.

Learning the values that features have for particular word senses must be done when new senses are created; and even for features imported from resources like the link grammar, the possibility of corrections must obviously be accepted. This kind of learning can be done by straightforward inference – inference from examples of word usage, and by analogy from features for similar words. A simple example to think about, e.g., is learning the verb sense of "fax" when only the noun sense is known.

Next, the learning of new features can be viewed as a reasoning problem, in that inference can learn new relations applied to nodes representing syntactic senses of words. In principle, these "features" may be very general or very specialized, depending on the case. New feature learning, in practice, requires a lot of examples, and is a more fundamental but less common kind of learning than learning feature values for known word senses. A good example would be the learning of "third person" by an agent that knows only first and second person.

In this example, it's clear that information from embodied experience would be extremely helpful. In principle, it could be learned from corpus analysis alone – but the presence of knowledge that certain words ("him", "her", "they", etc.) tend to occur in association with observed agents different from the speaker or the hearer, would certainly help a lot with identifying "third person" as a separate construct. It seems that either a very large number of un-embodied examples or a relatively small number of embodied examples would be needed to support the inference of the "third person" feature. And we suspect this example is typical – i.e. that the most effective route to new feature structure learning involves both embodied social experience and rather deep commonsense knowledge about the world.

44.15.3 Transformation and Semantic Mapping Rule Learning

Word sense learning and feature structure learning are important parts of language learning, but they're far from the whole story. An equally important role is played by *linguistic transformations*, such as the rules used in RelEx and RelEx2Frame. At least some of these must be learned based on experience, for human-level intelligent language processing to proceed.

Each of these transformations can be straightforwardly cast as an ImplicationLink between PredicateNodes, and hence formalistically can be learned

by PLN inference, combined with one or another heuristic methods for compound predicate creation. The question is what knowledge exists for PLN to draw on in assessing the strengths of these links, and more critically, to guide the heuristic predicate formation methods. This is a case that likely requires the full complexity of "integrative predicate learning" as discussed in Chapter 41. And, as with feature structure learning, it's a case that will be much more effectively handled using knowledge from social embodied experience alongside purely linguistic knowledge.

44.16 Experiential Language Learning

We have talked a great deal about "engineered" approaches to NL comprehension and only peripherally about experiential approaches. But there has been a not-so-secret plan underlying this approach. There are many approaches to experiential language learning, ranging from a "tabula rasa" approach in which language is just treated as raw data, to an approach where the whole structure of a language comprehension system is programmed in, and "merely" the *content* remains to be learned. There isn't much to say about the tabula rasa approach – we have already discussed CogPrime 's approach to learning, and in principle it is just as applicable to language learning as to any other kind of learning. The more structured approach has more unique aspects to it, so we will turn attention to it here. Of course, various intermediate approaches may be constructed by leaving out various structures.

The approach to experiential language learning we consider most promising is based on the PROWL approach, discussed above. In this approach one programs in a certain amount of "universal grammar," and then allows the system to learn content via experience that obeys this universal grammar. In a PROWL approach, the basic linguistic representational infrastructure is given by the Atomspace that already exists in OpenCog, so the content of "universal grammar" is basically

- the propensity to identify words
- the propensity to create a small set of asymmetric (i.e. parent/child) labeled relationship types, to use to label relationships between semantically related word-instances. These are "syntactic link types."
- the set of *constraints on syntactic links* implicit in word grammar, e.g. landmark transitivity

Building in the above items, without building in any particular syntactic links, seems enough to motivate a system to learn a *grammar resembling that of human languages*.

Of course, experiential language learning of this nature is very, very different from "tabula rasa" experiential language learning. But we note that,

while PROWL style experiential language learning seems like a *difficult* problem given existing AI technologies, tabula rasa language learning seems like a nearly unapproachable problem. One could infer from this that current AI technologies are simply inadequate to approach the problem that the young human child mind solves. However, there seems to be some solid evidence that the young human child mind does contain some form of universal grammar guiding its learning. Though we don't yet know what form this universal prior linguistic knowledge takes in the human mind or brain, the evidence regarding common structures arising spontaneously in various unrelated Creole languages is extremely compelling [Bic08], supporting ideas presented previously based on different lines of evidence. So we suggest that PROWL based experiential language learning is actually conceptually closer to human child language learning than a tabula rasa approach – although we certainly don't claim that the PROWL based approach builds in the exact same things as the human genome does.

What we need to make experiential language learning work, then, is a language-focused inference-control mechanism that includes, e.g.

- a propensity to look for syntactic link types, as outlined just above
- a propensity to form new word senses, as outlined earlier
- a propensity to search for implications of the general form of RelEx and RelEx2Frame rules

Given these propensities, it seems reasonable to expect a PLN inference system to be able to "fill in the linguistic content" based on its experience, using links between linguistic and other experiential content as its guide. This is a very difficult learning problem, to be sure, but it seems in principle a tractable one, since we have broken it down into a number of interrelated component learning problems in a manner guided by the structure of language.

Other aspects of language comprehension, such as word sense disambiguation and anaphor resolution, seem to plausibly follow from applying inference to linguistic data in the context of embodied experiential data, without requiring especial attention to inference control or supplying prior knowledge.

44.17 Which Path(s) Forward?

We have discussed a variety of approaches to achieving human-level NL comprehension in the CogPrime framework. Which approach do we think is best? All things considered, we suspect that a tabula rasa experiential approach is impractical, whereas a traditional computational linguistics approach (whether based on hand-coded rules, corpus analysis, or a combination thereof) will reach an intelligence ceiling well short of human capability. On the other hand we believe that all of these options

1. the creation of an engineered NL comprehension system (as we have already done), and the adaptation and enhancement of this system using learning that incorporates knowledge from embodied experience
2. the creation of an experiential learning based NL comprehension system using in-built structures, such as the PROWL based approach described above
3. the creation of an experiential learning based system as described above, using an engineered system (like the current one) as a "fitness estimation" resource in the manner described at the end of Chapter 43

have significant promise and are worthy of pursuit. Which of these approaches we focus on in our ongoing OpenCogPrime implementation work will depend on logistical issues as much as on theoretical preference.

Chapter 45
Natural Language Generation

Co-authored with Ruiting Lian and Rui Liu

45.1 Introduction

Language generation, unsurprisingly, shares most of the key features of language comprehension discussed in chapter 44 – after all, the division between generation and comprehension is to some extent an artificial convention, and the two functions are intimately bound up both in the human mind and in the CogPrime architecture.

In this chapter we discuss language generation, in a manner similar to the previous chapter's treatment of language comprehension. First we discuss our currently implemented, "engineered" language generation system, and then we discuss some alternative approaches:

- how a more experiential-learning based system might be made by retaining the basic structure of the engineered system but removing the "pre-wired" contents.
- how an "Atom2Link" system might be made, via reversing the Link2Atom system described in Chapter 44. This is the subject of implementation effort, at time of writing.

At the start of Chapter 44 we gave a high-level overview of a typical NL generation pipeline. Here we will focus largely but not entirely on the "syntactic and morphological realization" stage, which we refer to for simplicity as "sentence generation" (taking a slight terminological liberty, as "sentence fragment generation" is also included here). All of the stages of language generation are important, and there is a nontrivial amount of feedback among them. However, there is also a significant amount of autonomy, such that it often makes sense to analyze each one separately and then tease out its interactions with the other stages.

45.2 SegSim for Sentence Generation

The sentence generation approach currently taken in OpenCog (from 2009 - early 2012), which we call SegSim, is relatively simple and is depicted in Figure **??** and described as follows:

1. The NL generation system stores a large set of pairs of the form (semantic structure, syntactic/morphological realization)
2. When it is given a new semantic structure to express, it first breaks this semantic structure into natural parts, using a set of simple syntactic-semantic rules
3. For each of these parts, it then matches the parts against its memory to find relevant pairs (which may be full or partial matches), and uses these pairs to generate a set of syntactic realizations (which may be sentences or sentence fragments)
4. If the matching has failed, then (a) it returns to Step 2 and carries out the breakdown into parts again. But if this has happened too many times, then (b) it recourses to a different algorithm (most likely a search or optimization based approach, which is more computationally costly) to determine the syntactic realization of the part in question.
5. If the above step generated multiple fragments, they are pieced together, and a certain rating function is used to judge if this has been done adequately (using criteria of grammaticality and expected comprehensibility, among others). If this fails, then Step 3 is tried again on one or more of the parts; or Step 2 is tried again. (Note that one option for piecing the fragments together is to string together a number of different sentences; but this may not be judged optimal by the rating function.)
6. Finally, a "cleanup" phase is conducted, in which correct morphological forms are inserted, and articles and certain other "function words" are inserted.

The specific OpenCog software implementing the SegSim algorithm is called "NLGen"; this is an implementation of the SegSim concept that focuses on sentence generation from RelEx semantic relationship. In the current (early 2012) NLGen version, Step 1 is handled in a very simple way using a relational database; but this will be modified in future so as to properly use the AtomSpace. Work is currently underway to replace NLGen with a different "Atom2Link" approach, that will be described at the end of this chapter. But discussion of NLGen is still instructive regarding the intersection of language generation concepts with OpenCog concepts.

The substructure currently used in Step 2 is defined by the predicates of the sentence, i.e. we define one substructure for each predicate, which can be described as follows:

$$Predicate(Argument_i(Modify_j))$$

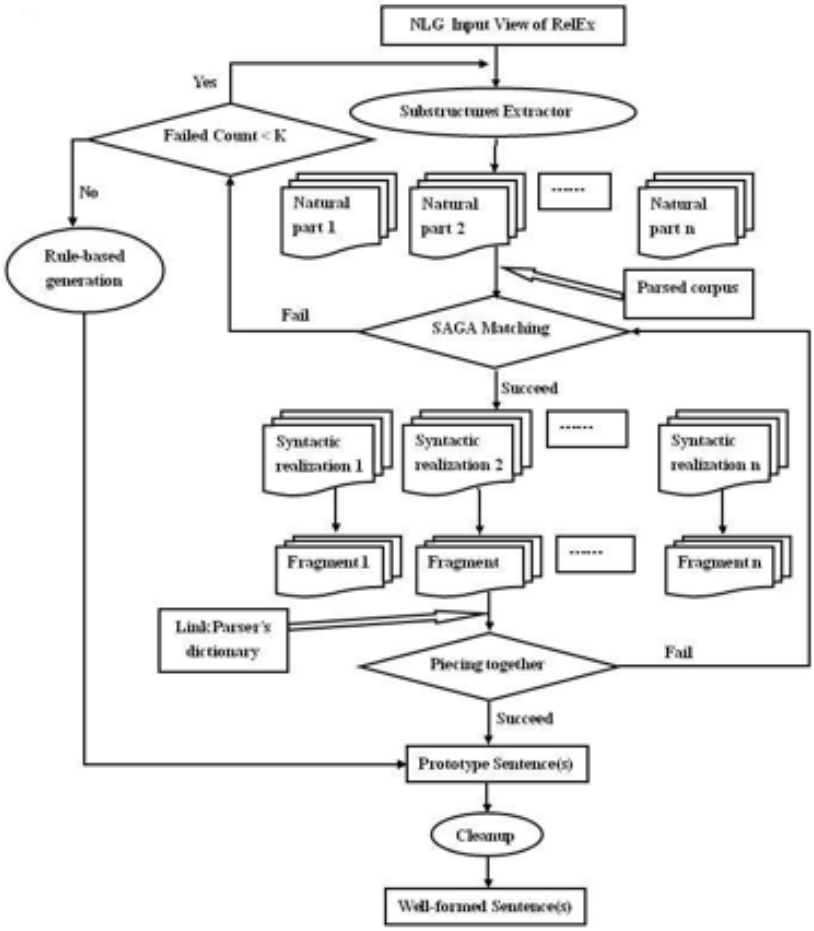


Fig. 45.1 A Overview of the SegSim Architecture for Language Generation

where

- $1 \leq i \leq m$ and $0 \leq j \leq n$ and m, n are integers
- "Predicate" stands for the predicate of the sentence, corresponding to the variable \$0 of the RelEx relationship _subj(\$0, \$1) or _obj(\$0, \$1)
- $Argument_i$ is the i-th semantic parameter related with the predicate
- $Modify_j$ is the j-th modifier of the $Argument_i$

If there is more than one predicate, then multiple subnets are extracted analogously.

For instance, given the sentence "I happily study beautiful mathematics in beautiful China with beautiful people." The substructure can be defined as Figure 45.2.

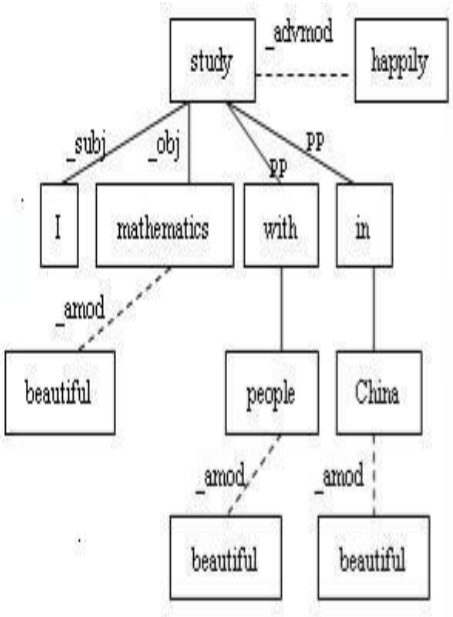


Fig. 45.2 Example of a substructure

For each of these substructures, Step 3 is supposed to match the substructures of a sentence against its global memory (which contains a large body of previously encountered [semantic structure, syntactic/morphological realization] pairs) to find the most similar or same substructures and the relevant syntactic relations to generate a set of syntactic realizations, which may be sentences or sentence fragments. In our current implementation, a customized subgraph matching algorithm has been used to match the subnets from the parsed corpus at this step.

If Step 3 generated multiple fragments, they must be pieced together. In Step 4, the Link Parser's dictionary has been used for detecting the dangling syntactic links corresponding to the fragments, which can be used to integrate the multiple fragments. For instance, in the example of Figure 45.3, according to the last 3 steps, SegSim would generate two fragments: "the parser will ignore the sentence" and "whose length is too long". Then it consults the Link Parser's dictionary, and finds that "whose" has a connector "Mr-", which is used for relative clauses involvi ng "whose", to connect to the previous noun "sentence". Analogously, we can integrate the other fragments into a whole sentence.

For instance, in the example of Figure 45.3, according to the last 3 steps, SegSim would generate two fragments: "the parser will ignore the sentence" and "whose length is too long". Then it consults the Link Parser's dictionary,

Fig. 45.3 Linkage of an example

and finds that "whose" has a connector "Mr-", which is used for relative clauses involving "whose", to connect to the previous noun "sentence". Analogously, we can integrate the other fragments into a whole sentence.

Finally, a "cleanup" or "post-processing" phase is conducted, applying the correct inflections to each word depending on the word properties provided by the input RelEx relations. For example, we can use the RelEx relation "DEFINITE-FLAG(cover, T)" to insert the article "the" in front of the word "cover". We have considered five factors in this version of NLGen: article, noun plural, verb intense, possessive and query type (the latter which is only for interrogative sentences).

In the "cleanup" step, we also use the chunk parser tool from OpenNLP* for adjusting the position of an article being inserted. For instance, consider the proto-sentence "I have big red apple." If we use the RelEx relation "noun_number(apple, singular)" to inflect the word "apple" directly, the final sentence will be "I have big red an apple", which is not well-formed. So we use the chunk parser to detect the phrase "big red apple" first, then apply the article rule in front of the noun phrase. This is a pragmatic approach which may be replaced with something more elegant and principled in later revisions of the NLGen system.

45.2.1 NLGen: Example Results

NLGen is currently in a relatively early stage of development, and does not handle the full range of linguistic and semantic phenomena that it will when it's completed. However, it can already express a variety of sentences encapsulating a variety of syntactic and semantic phenomena; in this section we will give some specific examples of what it can do.

The SegSim approach performs sentence generation by matching portions of propositional input to a large corpus of parsed sentences, therefore, when the successful matches can be found in the corpus, it can generate some similar and well-formed sentences via the relevant syntactic structures.

There currently do not exist any authoritative evaluation criteria for Natural Language Generation systems. Among many complicating factors here

* http://opennlp.sourceforge.net/

is the fact that different language generation systems have different kinds of inputs, depending on many things including their application area. So it's complicated to compare the results of NLGen with those obtained by other systems.

It is easier however to test whether NLGen is implementing SegSim successfully. One approach is to take a sentence, run it through RelEx to generate a set of relationships, and see if NLGen can regenerate the sentence from the relationship. We show here the results of some simple tests of this nature, performed with the current version of the system using a very small test corpus for similarity matching. Note: In each of these example results, the input of NLGen is the RelEx relationships produced by the sentence before "==>"; and the sentence to the right side of "==>" was one of the sentences generated by NLGen.

Example 1:

```
I study mathematics at Xiamen University.
==> I study mathematics at Xiamen\_University.
```

Example 2:

```
The red block is next to the blue cup.
==> The red block is next\_to the blue cup.
```

Example 3:

```
Ruiting walked from the table to the door.
==> Ruiting walked to the door from the table.
```

Example 4:

```
I intend to walk very slowly from Ben to Ruiting,
with Hugo's pencil in my hand.
==> With Hugo's pencil in my hand,
I intend to walk very slowly from Ben to Ruiting.
```

Example 5:

```
I'd like you to tell me where the red ball is.
==> I tell like you to 'd me where the red ball is.
```

Example 6:

```
I study at Xiamen University, which is located in China.
==> I study at Xiamen\_University.
```

Example 7:

```
What is next to the tree?
 ==> What is next to the tree?
```

Example 8:

```
Where  is the red ball?
==> Where is the red ball?
```

Example 9:

```
Pass me the ball.
 ==> Pass me the ball.
```

Example 10:

```
Tell Ben where the red ball is.
 ==> Tell Ben where the red ball is.
```

To make the process clearer, for Example 1 we also show the RelEx relationships produced from the sentence before the "==>":

For instance, in the example of Figure 45.3, according to the last 3 steps, SegSim would generate two fragments: "the parser will ignore the sentence" and "whose length is too long". Then it consults the Link Parser's dictionary, and finds that "whose" has a connector "Mr-", which is used for relative clauses involving "whose", to connect to the previous noun "sentence". Analogously, we can integrate the other fragments into a whole sentence.

```
pos(., punctuation)
noun_number(Xiamen_University, singular)
pos(Xiamen_University, noun)
DEFINITE-FLAG(Xiamen_University, T)
pos(at, prep)
_subj(study, I)
at(study, Xiamen_University)
tense(study, present)
pos(study, verb)
gender(I, person)
noun_number(I, singular)
pos(I, noun)
PRONOUN-FLAG(I, T)
DEFINITE-FLAG(I, T)
```

Fig. 45.4 RelEx relationships for Example 1

Figure 45.4 shows the relationships of Example 1 fed to NLGen as input. The types of the semantic relationships are documented in the RelEx's wiki pages[†].

These examples illustrate some key points about the current version of NLGen. It works well on simple, commonplace sentences (Example 1, 2), though it may reorder the sentence fragments sometimes (Example 3, 4). On the other hand, because of its reliance on matching against a corpus, NLGen is incapable of forming good sentences with syntactic structures not found in the corpus (Example 5, 6). On a larger corpus these examples would have given successful results. In Example 5, the odd error is due to the presence of too many "_subj" RelEx relationships in the relationship-set corresponding to the sentence, which distracts the matching process when it attempts to find similar substructures in the small test corpus. Then from Example 7 to 10, we can see NLGen still works well for question sentences and imperative sentence if the substructures we extract can be matched, but the substructures may be similar with the assertive sentence, so we need to refine it in the "clearupcleanup" step. For example: the substructures we extracted for the sentence "are you a student?" are the same as the ones for "you are a student?", since the two sentences both have the same binary RelEx relationships:

```
\_subj(be, you)

\_obj(be, student)
```

which are used to guide the extraction of the substructures. So we need to refine the sentence via some grammatical rules in the post-processing phase, using the word properties from RelEx, like "TRUTH-QUERY-FLAG(be, T)" which means if that the referent "be" is a verb/event and the event is involved is a question.

The particular shortcomings demonstrated in these examples are simple to remedy within the current NLGen framework, via simply expanding the corpus. However, to get truly general behavior from NLGen it will be necessary to insert some other generation method to cover those cases where similarity matching fails, as discussed above. The NLGen2 system created by Blake Lemoine [Lem10] is one possibility in this regard: based on RelEx and the link parser, it carries out rule-based generation using an implementation of Chomsky's Merge operator. Integration of NLGen with NLGen2 is currently being considered. We note that the Merge operator is computationally inefficient by nature, so that it will likely never be suitable for the primary sentence generation method in a language generation system. However, pairing NLGen for generation of familiar and routine utterances with a Merge-based approach for generation of complex or unfamiliar utterances, may prove a robust approach.

[†] http://opencog.org/wiki/RelEx#Relations_and_Features

45.3 Experiential Learning of Language Generation

As in the case of language comprehension, there are multiple ways to create an experiential learning based language generation system, involving various levels of "wired in" knowledge. Our best guess is that for generation as for comprehension, a "tabula rasa" approach will prove computationally intractable for quite some time to come, and an approach in which some basic structures and processes are provided, and then filled out with content learned via experience, will provide the greatest odds of success.

A highly abstracted version of SegSim may be formulated as follows:

1. The AI system stores semantic and syntactic structures, and its control mechanism is biased to search for, and remember, linkages between them
2. When it is given a new semantic structure to express, it first breaks this semantic structure into natural parts, using inference based on whatever implications it has in its memory that will serve this purpose
3. Its inference control mechanism is biased to carry out inferences with the following implication: For each of these parts, match it against its memory to find relevant pairs (which may be full or partial matches), and use these pairs to generate a set of syntactic realizations (which may be sentences or sentence fragments)
4. If the matching has failed to yield results with sufficient confidence, then (a) it returns to Step 2 and carries out the breakdown into parts again. But if this has happened too many times, then (b) it uses its ordinary inference control routine to try to determine the syntactic realization of the part in question.
5. If the above step generated multiple fragments, they are pieced together, and an attempt is made to infer, based on experience, whether the result will be effectively communicative. If this fails, then Step 3 is tried again on one or more of the parts; or Step 2 is tried again.
6. Other inference-driven transformations may occur at any step of the process, but are particularly likely to occur at the end. In some languages these transformations may result in the insertion of correct morphological forms or other "function words."

What we suggest is that it may be interesting to supply a CogPrime system with this overall process, and let it fill in the rest by experiential adaptation. In the case that the system is learning to comprehend at the same time as it's learning to generate, this means that its early-stage generations will be based on its rough, early-stage comprehension of syntax – but that's OK. Comprehension and generation will then "grow up" together.

45.4 Atom2Link

A subject of current research is the extension of the Link2Atom approach mentioned above into a reverse-order, Atom2Link system for language generation.

Given that the Link2Atom rules are expressed as ImplicationLinks, they can be reversed automatically and immediately – although, the reversed versions will not necessarily have the same truth values. So if a collection of Link2Atom rules are learned from a corpus, then they can be used to automatically generate a set of Atom2Link rules, each tagged with a probabilistic truth value. Application of the whole set of Atom2Link rules to a given Atomset in need of articulation, will result in a collection of link-parse links.

To produce a sentence from such a collection of link-parse links, another process is also needed, which will select a subset of the collection that corresponds to a complete sentence, legally parse-able via the link parser. The overall collection might naturally break down into more than one sentence.

In terms of the abstracted version of SegSim given above, the primary difference between NLGen and SegSim lies in Step 3. Link2Atom replaces the SegSim "data-store matching" algorithm with inference based on implications obtained from reversing the implications used for language comprehension.

45.5 Conclusion

There are many different ways to do language generation within OpenCog, ranging from pure experiential learning to a database-driven approach like NLGen. Each of these different ways may have value for certain applications, and it's unclear which ones may be viable in a human-level AGI context. Conceptually we would favor a pure experiential learning approach, but, we are currently exploring a "compromise" approach based on Atom2Link. This is an area where experimentation is going to tell us more than abstract theory.

Chapter 46
Embodied Language Processing

Co-authored with Samir Araujo and Welter Silva

46.1 Introduction

"Language" is an important abstraction – but one should never forget that it's an abstraction. Language evolved in the context of embodied action, and even the most abstract language is full of words and phrases referring to embodied experience. Even our mathematics is heavily based on our embodied experience – geometry is about space; calculus is about space and time; algebra is a sort of linguistic manipulation generalized from experience-oriented language, etc. (see [**?**] for detailed arguments in this regard). To consider language in the context of human-like general intelligence, one needs to consider it in the context of embodied experience.

There is a large literature on the importance of embodiment for child language learning, but perhaps the most eloquent case has been made by Michael Tomasello, in his excellent book *Constructing a Language* **??**. Citing a host of relevant research by himself and others, Tomasello gives a very clear summary of the value of social interaction and embodiment for language learning in human children. And while he doesn't phrase it in these terms, the picture he portrays includes central roles for reinforcement, imitative and corrective learning. Imitative learning is obvious: so much of embodied language learning has to do with the learner copying what it has heard other say in similar contexts. Corrective learning occurs every time a parent rephrases something for a child.

In this chapter, after some theoretical discussion of the nature of symbolism and the role of gesture and sound in language, we describe some computational experiments run with OpenCog controlling virtual pets in a virtual world, regarding the use of embodied experience for anaphor resolution and question-answering. These comprise an extremely simplistic example of the interplay between language and embodiment, but have the advantage of concreteness, since they were actually implemented and experimented with. Some of the specific OpenCog tools used in these experiments are no longer

current (e.g. the use of RelEx2Frame, which is now deprecated in favor of alternative approaches to mapping parses into more abstract semantic relationships); but the basic principles and flow illustrated here are still relevant to current and future work.

46.2 Semiosis

The foundation of communication is semiosis – the representation between the signifier and the signified. Often the signified has to do with the external world or the communicating agent's body; hence the critical role of embodiment in language.

Thus, before turning to the topic of embodied *language* use and learning per se, we will briefly treat the related topic of how an AGI system may learn *semiosis* itself via its embodied experience. This is a large and rich topic, but we will restrict ourselves to giving a few relatively simple examples intended to make the principles clear. We will structure our discussion of semiotic learning according to Charles Sanders Peirce's theory of semiosis [Pei34], in which there are three basic types of signs: icons, indices and symbols.

In Peirce's ontology of semiosis, an icon is a sign that physically resembles what it stands for. Representational pictures, for example, are icons because they look like the thing they represent. Onomatopoeic words are icons, as they sound like the object or fact they signify. The iconicity of an icon need not be immediate to appreciate. The fact that "kirikiriki" is iconic for a rooster's crow is not obvious to English-speakers yet it is to many Spanish-speakers; and the the converse is true for "cock-a-doodle-doo."

Next, an index is a sign whose occurrence probabilistically implies the occurrence of some other event or object (for reasons other than the habitual usage of the sign in connection with the event or object among some community of communicating agents). The index can be the cause of the signified thing, or its consequence, or merely be correlated to it. For example, a smile on your face is an index of your happy state of mind. Loud music and the sound of many people moving and talking in a room is an index for a party in the room. On the whole, more contextual background knowledge is required to appreciate an index than an icon.

Finally, any sign that is not an icon or index is a symbol. More explicitly, one may say that a symbol is a sign whose relation to the signified thing is conventional or arbitrary. For instance, the stop sign is a symbol for the imperative to stop; the word "dog" is a symbol for the concept it refers to.

The distinction between the various types of signs is not always obvious, and some signs may have multiple aspects. For instance, the thumbs-up gesture is a symbol for positive emotion or encouragement. It is not an index – unlike a smile which is an index for happiness because smiling is intrinsically biologically tied to happiness, there is no intrinsic connection between

the thumbs-up signal and positive emotion or encouragement. On the other hand, one might argue that the thumbs-up signal is very weakly iconic , in that its up-ness resembles the subjective up-ness of a positive emotion (note that in English an idiom for happiness is "feeling up").

Teaching an embodied virtual agent to recognize simple icons is a relatively straightforward learning task. For instance, suppose one wanted to teach an agent that in order to get the teacher to give it a certain type of object, it should go to a box full of pictures and select a picture of an object of that type, and bring it to the teacher. One way this may occur in an OpenCog-controlled agent is for the agent to learn a rule of the following form:

```
ImplicationLink
    ANDLink
        ContextLink
             Visual
            SimilarityLink $X $Y
    PredictiveImplicationLink
        SequentialANDLink
            ExecutionLink goto box
            ExecutionLink grab $X
            ExecutionLink goto teacher
        EvaluationLink give me teacher $Y
```

While not a trivial learning problem, this is straightforward to a CogPrime -controlled agent that is primed to consider visual similarities as significant (i.e. is primed to consider the visual-appearance context within its search for patterns in its experience).

Next, proceeding from icons to indices: Suppose one wanted to teach an agent that in order to get the teacher to give it a certain type of object, it should go to a box full of pictures and select a picture of an object that has commonly been used together with objects of that type, and bring it to the teacher. This is a combination of iconic and indexical semiosis, and would be achieved via the agent learning a rule of the form

```
Implication
   AND
      Context
          Visual
          Similarity $X $Z
       Context
           Experience
           SpatioTemporalAssociation $Z $Y
       PredictiveImplication
           SequentialAnd
              Execution goto box
              Execution grab $X
              Execution goto teacher
           Evaluation give me teacher $Y
```

Symbolism, finally, may be seen to emerge as a fairly straightforward extension of indexing. After all, how does an agent come to learn that a certain symbol refers to a certain entity? An advanced linguistic agent can learn this via explicit verbal instruction, e.g. one may tell it "The word Ôhideous' means Ôvery ugly'." But in the early stages of language learning, this sort of instructional device is not available, and so the way an agent learns that a word is associated with an object or an action is through spatiotemporal association. For instance, suppose the teacher wants to teach the agent to dance every time the teacher says the word "dance" – a very simple example of symbolism. Assuming the agent already knows how to dance, this merely requires the agent learn the implication

```
PredictiveImplication
   SequentialAND
      Evaluation say teacher me "dance"
      Execution dance
   give teacher me Reward
```

And, once this has been learned, then simultaneously the relationship

```
SpatioTemporalAssociation dance "dance"
```

will be learned. What's interesting is what happens after a number of associations of this nature have been learned. Then, the system may infer a general rule of the form

```
Implication
   AND
      SpatioTemporalAssociation \$X \$Z
      HasType \$X GroundedSchema
   PredictiveImplication
      SequentialAND
         Evaluation say teacher me \$Z
         Execution \$X
      Evaluation give teacher me Reward
```

This implication represents the general rule that if the teacher says a word corresponding to an action the agent knows how to do, and the agent does it, then the agent may get a reward from the teacher. Abstracting this from a number of pertinent examples is a relatively straightforward feat of probabilistic inference for the PLN inference engine.

Of course, the above implication is overly simplistic, and would lead an agent to stupidly start walking every time its teacher used the word "walk" in conversation and the agent overheard it. To be useful in a realistic social context, the implication must be made more complex so as to include some of the pragmatic surround in which the teacher utters the word or phrase $Z.

46.3 Teaching Gestural Communication

Based on the ideas described above, it is relatively straightforward to teach virtually embodied agents the elements of gestural comunication. This is important for two reasons: gestural communication is extremely useful unto itself, as one sees from its role in communication among young children and primates [22]; and, gestural communication forms a foundation for verbal communication, during the typical course of human language learning [23]. Note for instance the study described in [22], which "reports empirical longitudinal data on the early stages of language development," concluding that

...the output systems of speech and gesture may draw on underlying brain mechanisms common to both language and motor functions. We analyze the spontaneous interaction with their parents of three typically-developing children (2 M, 1 F) videotaped monthly at home between 10 and 23 months of age. Data analyses focused on the production of actions, representational and deictic gestures and words, and gesture-word combinations. Results indicate that there is a continuity between the production of the first action schemes, the first gestures and the first words produced by children. The relationship between gestures and words changes over time. The onset of two-word speech was preceded by the emergence of gesture-word combinations.

If young children learn language as a continuous outgrowth of gestural communication, perhaps the same approach may be effective for (virtually or physically) embodied AI's.

An example of an iconic gesture occurs when one smiles explicitly to illustrate to some other agent that one is happy. Smiling is a natural expression of happiness, but of course one doesn't always smile when one's happy. The reason that explicit smiling is iconic is that the explicit smile actually resembles the unintentional smile, which is what it "stands for."

This kind of iconic gesture may emerge in a socially-embedded learning agent through a very simple logic. Suppose that when the agent is happy, it benefits from its nearby friends being happy as well, so that they may then do happy things together. And suppose that the agent has noticed that when it smiles, this has a statistical tendency to make its friends happy. Then, when it is happy and near its friends, it will have a good reason to smile. So through very simple probabilistic reasoning, the use of explicit smiling as a communicative tool may result. But what if the agent is not actually happy, but still wants some other agent to be happy? Using the reasoning from the prior paragraph, it will likely figure out to smile to make the other agent happy – even though it isn't actually happy.

Another simple example of an iconic gesture would be moving one's hands towards one's mouth, mimicking the movements of feeding oneself, when one wants to eat. Many analogous iconic gestures exist, such as doing a small solo part of a two-person dance to indicate that one wants to do the whole dance together with another person. The general rule an agent needs to learn in order to generate iconic gestures of this nature that, in the context of shared

activity, mimicking part of a process will sometimes serve the function of evoking that whole process.

This sort of iconic gesture may be learned in essentially the same way as an indexical gesture such as a dog repeatedly drawing the owner's attention to the owner's backpack, when the dog wants to go outside. The dog doesn't actually care about going outside with the backpack – he would just as soon go outside without it – but he knows the backpack is correlated with going outside, which is his actual interest.

The general rule here is

```
R :=
Implication
SimultaneousImplication
Execution \$X
\$Y
PredictiveImplication
\$X
\$Y
```

I.e., if doing $X often correlates with $Y, then maybe doing $X will bring about $Y. This sort of rule can bring about a lot of silly "superstitious" behavior but also can be particularly effective in social contexts, meaning in formal terms that

```
Context
near\_teacher
R
```

holds with a higher truth value than R itself. This is a very small conglomeration of semantic nodes and links yet it encapsulates a very important communicational pattern: that if you want something to happen, and act out part of it – or something historically associated with it – around your teacher, then the thing may happen.

Many other cases of iconic gesture are more complex and mix iconic with symbolic aspects. For instance, one waves one hand away from oneself, to try to get someone else to go away. The hand is moving, roughly speaking, in the direction one wants the other to move in. However, understanding the meaning of this gesture requires a bit of savvy or experience. One one does grasp it, however, then one can understand its nuances: For instance, if I wave my hand in an arc leading from your direction toward the direction of the door, maybe that means I want you to go out the door.

Purely symbolic (or nearly so) gestures include the thumbs-up symbol mentioned above, and many others including valence-indicating symbols like a nodded head for YES, a shaken-side-to-side head for NO, and shrugged shoulders for "I don't know." Each of these valence-indicating symbols actually indicates a fairly complex concept, which is learned from experience partly via attention to the symbol itself. So, an agent may learn that the

nodded head corresponds with situations where the teacher gives it a reward, and also with situations where the agent makes a request and the teacher complies. The cluster of situations corresponding to the nodded-head then forms the agent's initial concept of "positive valence," which encompasses, loosely speaking, both the good and the true.

Summarizing our discussion of gestural communication: An awful lot of language exists between intelligent agents even if no word is ever spoken. And, our belief is that these sorts of non-verbal semiosis form the best possible context for the learning of verbal language, and that to attack verbal language learning outside this sort of context is to make an intrinsically-difficult problem even harder than it has to be. And this leads us to the final part of the paper, which is a bit more speculative and adventuresome. The material in this section and the prior ones describes experiments of the sort we are currently carrying out with our virtual agent control software. We have not yet demonstrated all the forms of semiosis and non-linguistic communication described in the last section using our virtual agent control system, but we have demonstrated some of them and are actively working on extending our system's capabilities. In the following section, we venture a bit further into the realm of hypothesis and describe some functionalities that are beyond the scope of our current virtual agent control software, but that we hope to put into place gradually during the next 1-2 years. The basic goal of this work is to move from non-verbal to verbal communication.

It is interesting to enumerate the aspects in which each of the above components appears to be capable of tractable adaptation via experiential, embodied learning:

- Words and phrases that are found to be systematically associated with particular objects in the world, may be added to the "gazeteer list" used by the entity extractor
- The link parser dictionary may be automatically extended. In cases where the agent hears a sentence that is supposed to describe a certain situation, and realizes that in order for the sentence to be mapped into a set of logical relationships accurately describing the situation, it would be necessary for a certain word to have a certain syntactic link that it doesn't have, then the link parser dictionary may be modified to add the link to the word. (On the other hand, creating new link parser link types seems like a very difficult sort of learning – not to say it is unaddressable, but it will not be our focus in the near term.)
- Similar to with the link parser dictionary, if it is apparent that to interpret an utterance in accordance with reality a RelEx rule must be added or modified, this may be automatically done. The RelEx rules are expressed in the format of relatively simple logical implications between Boolean combinations of syntactic and semantic relationships, so that learning and modifying them is within the scope of a probabilistic logic system such as Novamente's PLN inference engine.

- The rules used by RelEx2Frame may be experientially modified quite analogously to those used by RelEx
- Our current statistical parse ranker ranks an interpretation of a sentence based on the frequency of occurrence of its component links across a parsed corpus. A deeper approach, however, would be to rank an interpretation based on its commonsensical plausibility, as inferred from experienced-world-knowledge as well as corpus-derived knowledge. Again, this is within the scope of what an inference engine such as PLN should be able to do.
- Our word sense disambiguation and reference resolution algorithms involve probabilistic estimations that could be extended to refer to the experienced world as well as to a parsed corpus. For example, in assessing which sense of the noun "run" is intended in a certain context, the system could check whether stockings, or sports-events or series-of-events, are more prominent in the currently-observed situation. In assessing the sentence "The children kicked the dogs, and then they laughed," the system could map "they" into "children" via experientially-acquired knowledge that children laugh much more often than dogs.
- NLGen uses the link parser dictionary, treated above, and also uses rules analogous to (but inverse to) RelEx rules, mapping semantic relations into brief word-sequences. The "gold standard" for NLGen is whether, when it produces a sentence S from a set R of semantic relationships, the feeding of S into the language comprehension subsystem produces R (or a close approximation) as output. Thus, as the semantic mapping rules in RelEx and RelEx2Frame adapt to experience, the rules used in NLGen must adapt accordingly, which poses an inference problem unto itself.

All in all, when one delves in detail into the components that make up our hybrid statistical/rule-based NLP system, one sees there is a strong opportunity for experiential adaptive learning to substantially modify nearly every aspect of the NLP system, while leaving the basic framework intact.

This approach, we suggest, may provide means of dealing with a number of problems that have systematically vexed existing linguistic approaches. One example is parse ranking for complex sentences: this seems almost entirely a matter of the ability to assess the semantic plausibility of different parses, and doing this based on statistical corpus analysis seems unreasonable. One needs knowledge about a world to ground reasoning about plausiblity.

Another example is preposition disambiguation, a topic that is barely dealt with at all in the computational linguistics literature (see e.g. [33] for an indication of the state of the art). Consider the problem of assessing which meaning of "with" is intended in sentences like "I ate dinner with a fork", "I ate dinner with my sister", "I ate dinner with dessert." In performing this sort of judgment, an embodied system may use knowledge about which interpretations have matched observed reality in the case of similar utterances it has processed in the past, and for which it has directly seen the situations referred to by the utterances. If it has seen in the past, through direct embodied

experience, that when someone said "I ate cereal with a spoon," they meant that the spoon was their tool not part of their food or their eating-partner; then when it hears "I ate dinner with a fork," it may match "cereal" to "dinner" and "spoon" to "fork" (based on probabilistic similarity measurement) and infer that the interpretation of "with" in the latter sentence should also be to denote a tool. How does this approach to computational language understanding tie in with gestural and general semiotic learning as we discussed earlier? The study of child language has shown that early language use is not purely verbal by any means, but is in fact a complex combination of verbal and gestural communication [23]. With the exception of point 1 (entity extraction) above, every one of our instances of experiential modification of our language framework listed above involves the use of an understanding of what situation actually exists in the world, to help the system identify what the logical relationships output by the NLP system are supposed to be in a certain context. But a large amount of early-stage linguistic communication is social in nature, and a large amount of the remainder has to do with the body's relationship to physical objects. And, in understanding "what actually exists in the world" regarding social and physical relationships, a full understanding of gestural communication is important. So, the overall pathway we propose for achieving robust, ultimately human-level NLP functionality is as follows:

- The capability for learning diverse instances of semiosis is established
- Gestural communication is mastered, via nonverbal imitative/reinforcement/corrective learning mechanisms such as we are now utilizing for our embodied virtual agents
- Gestural communication, combined with observation of and action in the world and verbal interaction with teachers, allows the system to adapt numerous aspects of its initial NLP engine to allow it to more effectively interpret simple sentences pertaining to social and physical relationships
- Finally, given the ability to effectively interpret and produce these simple and practical sentences, probabilistic logical inference allows the system to gradually extend this ability to more and more complex and abstract senses, incrementally adapting aspects of the NLP engine as its scope broadens.

In this brief section we will mention another potentially important factor that we have intentionally omitted in the above analysis – but that may wind up being very important, and that can certainly be taken into account in our framework if this proves necessary. We have argued that gesture is an important predecessor to language in human children, and that incorporating it in AI language learning may be valuable. But there is another aspect of early language use that plays a similar role to gesture, which we have left out in the above discussion: this is the acoustic aspects of speech.

Clearly, pre-linguistic children make ample use of communicative sounds of various sorts. These sounds may be iconic, indexical or symbolic; and they

may have a great deal of subtlety. Steven Mithen [34] has argued that non-verbal utterances constitute a kind of proto-language, and that both music and language evolved out of this. Their role in language learning is well-documented also [35]. We are uncertain as to whether an exclusive focus on text rather than speech would critically impair the language learning process of an AI system. We are fairly strongly convinced of the importance of gesture because it seems bound up with the importance of semiosis – gesture, it seems, is how young children learn flexible semiotic communication skills, and then these skills are gradually ported from the gestural to the verbal domain. Semiotically, on the other hand, phonology doesn't seem to give anything special beyond what gesture gives. What it does give is an added subtlety of emotional expressiveness – something that is largely missing from virtual agents as implemented today, due to the lack of really fine-grained facial expressions. Also, it provides valuable clues to parsing, in that groups of words that are syntactically bound together are often phrased together acoustically.

If one wished to incorporate acoustics into the framework described above, it would not be objectionably difficult on a technical level. Speech-to-text [36] and text-to-speech software [37] both exist, but neither have been developed with a view specifically toward conveyance of emotional information. One could approach the problem of assessing the emotional state of an utterance based on its sound as a supervised categorization problem, to be solved via supplying a machine learning algorithm with training data consisting of human-created pairs of the form (utterance, emotional valence). Similarly, one could tune the dependence of text-to-speech software for appropriate emotional expressiveness based on the same training corpus.

46.4 Simple Experiments with Embodiment and Anaphor Resolution

Now we turn to some fairly simple practical work that was done in 2008 with the OpenCog -based PetBrain software, involving the use of virtually embodied experience to help with interpretation of linguistic utterances. This work has been superseded somewhat by more recent work using OpenCog to control virtual agents; but the PetBrain work was especially clear and simple, so suitable in an expository sense for in-depth discussion here.

One of the two ways the PetBrain related language processing to embodied experience was via using the latter to resolve anaphoric references in text produced by human-controlled avatars.

The PetBrain controlled agent lived in a world with many objects, each one with their own characteristics. For example, we could have multiple balls, with varying colors and sizes. We represent this in the OpenCog Atomspace via using multiple nodes: a single ConceptNode to represent the concept

"ball", a WordNode associated with the word "ball", and numerous SemeNodes representing particular balls. There may of course also be ConceptNodes representing ball-related ideas not summarized in any natural language word, e.g. "big fat squishy balls," "balls that can usefully be hit with a bat", etc.

As the agent interacts with the world, it acquires information about the objects it finds, through perceptions. The perceptions associated with a given object are stored as other nodes linked to the node representing the specific object instance. All this information is represented in the Atomspace using FrameNet-style relationships (exemplified in the next section).

When the user says, e.g., "Grab the red ball", the agent needs to figure out which specific ball the user is referring to – i.e. it needs to invoke the Reference Resolution (RR) process. RR uses the information in the sentence to select instances and also a few heuristic rules. Broadly speaking, Reference Resolution maps nouns in the user's sentences to actual objects in the virtual world, based on world-knowledge obtained by the agent through perceptions.

In this example, first the brain selects the ConceptNodes related to the word "ball". Then it examines all individual instances associated with these concepts, using the determiners in the sentence along with other appropriate restrictions (in this example the determiner is the adjective "red"; and since the verb is "grab" it also looks for objects that can be fetched). If it finds more than one "fetchable red ball", a heuristic is used to select one (in this case, it chooses the nearest instance).

The agent also needs to map pronouns in the sentences to actual objects in the virtual world. For example, if the user says "I like the red ball. Grab it," the agent must map the pronoun "it" to a specific red ball. This process is done in two stages: first using anaphor resolution to associate the pronoun "it" with the previously heard noun "ball"; then using reference resolution to associate the noun "ball" with the actual object.

The subtlety of anaphor resolution is that there may be more than one plausible "candidate" noun corresponding to a given pronouns. As noted above, at time writing RelEx's anaphor resolution system is somewhat simplistic and is based on the classical Hobbs algorithm[Hob78]. Basically, when a pronoun (it, he, she, they and so on) is identified in a sentence, the Hobbs algorithm searches through recent sentences to find the nouns that fit this pronoun according to number, gender and other characteristics. The Hobbs algorithm is used to create a ranking of candidate nouns, ordered by time (most recently mentioned nouns come first).

We improved the Hobbs algorithm results by using the agent's world-knowledge to help choose the best candidate noun. Suppose the agent heard the sentences:

```
``The ball is red.''
``The stick is brown.''
```

and then it receives a third sentence

```
``Grab it.''.
```

the anaphor resolver will build a list containing two options for the pronoun "it" of the third sentence: ball and stick. Given that the stick corresponds to the most recently mentioned noun, the agent will grab it instead of (as Hobbs would suggest) the ball.

Similarly, if the agent's history contains

```
``From here I can see as tree and a ball.''
``Grab it.''
```

Hobbs algorithm returns as candidate nouns "tree" and "ball", in this order. But using our integrative Reference Resolution process, the agent will conclude that a tree cannot be grabbed, so this candidate is discarded and "ball" is chosen.

46.5 Simple Experiments with Embodiment and Question Answering

The PetBrain was also capable of answering simple questions about its feelings/emotions (happiness, fear, etc.) and about the environment in which it lives. After a question is asked to the agent, it is parsed by RelEx and classified as either a truth question or a discursive one. After that, RelEx rewrites the given question as a list of Frames (based on FrameNet* with some customizations), which represent its semantic content. The Frames version of the question is then processed by the agent and the answer is also written in Frames. The answer Frames are then sent to a module that converts it back to the RelEx format. Finally the answer, in RelEx format, is processed by the NLGen module, that generates the text of the answer in English. We will discuss this process here in the context of the simple question "What is next to the tree?", which in an appropriate environment receives the answer "The red ball is next to the tree."

Question answering (QA) of course has a long history in AI [May04], and our approach fits squarely into the tradition of "deep semantic QA systems"; however it is innovative in its combination of dependency parsing with FrameNet and most importantly in the manner of its integration of QA with an overall cognitive architecture for agent control.

46.5.1 Preparing/Matching Frames

In order to answer an incoming question, the agent tries to match the Frames list, created by RelEx, against the Frames stored in its own memory. In gen-

* http://framenet.icsi.berkeley.edu

eral these Frames could come from a variety of sources, including inference, concept creation and perception; but in the current PetBrain they primarily come from perception, and simple transformations of perceptions.

However, the agent cannot use the incoming perceptual Frames in their original format because they lack grounding information (information that connects the mentioned elements to the real elements of the environment). So, two steps are then executed before trying to match the Frames: Reference Resolution (described above) and Frames Rewriting. Frames Rewriting is a process that changes the values of the incoming Frames elements into grounded values. Here is an example:

Incoming Frame (Generated by RelEx)

```
EvaluationLink
  DefinedFrameElementNode Color:Color
  WordInstanceNode ``red@aaa''
EvaluationLink
  DefinedFrameElementNode Color:Entity
  WordInstanceNode ``ball@bbb''
ReferenceLink
  WordInstanceNode ``red@aaa''
  WordNode ``red''
```

After Reference Resolution

```
ReferenceLink
  WordInstanceNode ``ball@bbb''
  SemeNode ``ball_99''
```

Grounded Frame (After Rewriting)

```
EvaluationLink
  DefinedFrameElementNode Color:Color
  ConceptNode ``red''
EvaluationLink
  DefinedFrameElementNode Color:Entity
  SemeNode ``ball_99''
```

Frame Rewriting serves to convert the incoming Frames to the same structure used by the Frames stored into the agent's memory. After Rewriting, the new Frames are then matched against the agent's memory and if all Frames were found in it, the answer is known by the agent, otherwise it is unknown.

In the PetBrain system, if a truth question was posed and all Frames were matched successfully, the answer would be be "yes"; otherwise the answer is "no". Mapping of ambiguous matching results into ambiguous responses were not handled in the PetBrain.

If the question requires a discursive answer the process is slightly different. For known answers the matched Frames are converted into RelEx format by

Frames2RelEx and then sent to NLGen, which prepares the final English text to be answered. There are two types of unknown answers. The first one is when at least one Frame cannot be matched against the agent's memory and the answer is "I don't know". And the second type of unknown answer occurs when all Frames were matched successfully but they cannot be correctly converted into RelEx format or NLGen cannot identify the incoming relations. In this case the answer is "I know the answer, but I don't know how to say it".

46.5.2 Frames2RelEx

As mentioned above, this module is responsible for receiving a list of grounded Frames and returning another list containing the relations, in RelEx format, which represents the grammatical form of the sentence described by the given Frames. That is, the Frames list represents a sentence that the agent wants to say to another agent. NLGen needs an input in RelEx Format in order to generate an English version of the sentence; Frames2RelEx does this conversion.

Currently, Frames2RelEx is implemented as a rule-based system in which the preconditions are the required frames and the output is one or more RelEx relations e.g.

```
#Color(Entity,Color) =>
  present($2) .a($2) adj($2) _predadj($1, $2)
    definite($1) .n($1) noun($1) singular($1)
      .v(be) verb(be) punctuation(.) det(the)
```

where the precondition comes before the symbol => and *Color* is a frame which has two elements: Entity and Color. Each element is interpreted as a variable $Entity = \$1$ and $Color = \$2$. The effect, or output of the rule, is a list of RelEx relations. As in the case of RelEx2Frame, the use of hand-coded rules is considered a stopgap, and for a powerful AGI system based on this framework such rules will need to be learned via experience.

46.5.3 Example of the Question Answering Pipeline

Turning to the example "What is next to the tree?", Figure 46.5.3 illustrates the processes involved:

The question is parsed by RelEx, which creates the frames indicating that the sentence is a question regarding a location reference (next) relative to an object (tree). The frame that represents questions is called Questioning and

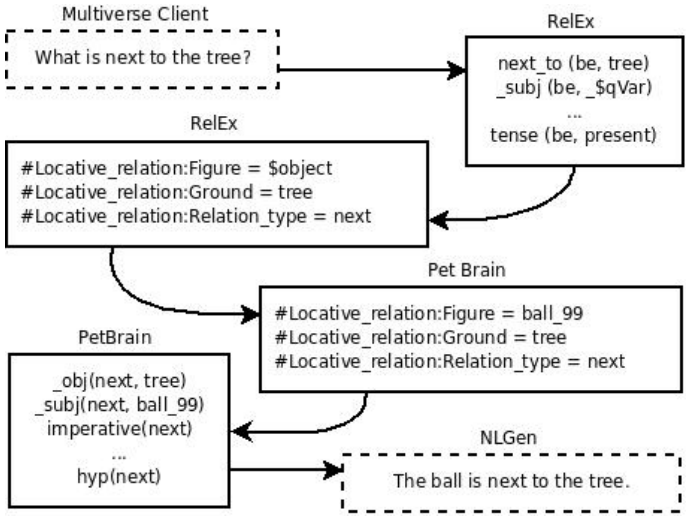


Fig. 46.1 Overview of current PetBrain language comprehension process

it contains the elements Manner that indicates the kind of question (truth-question, what, where, and so on), Message that indicates the main term of the question and Addressee that indicates the target of the question. To indicate that the question is related to a location, the *Locative_relation* frame is also created with a variable inserted in its element Figure, which represents the expected answer (in this specific case, the object that is next to the tree).

The question-answer module tries to match the question frames in the Atomspace to fit the variable element. Suppose that the object that is next to the tree is the red ball. In this way, the module will match all the frames requested and realize that the answer is the value of the element Figure of the frame *Locative_relation* stored in the Atom Table. Then, it creates location frames indicating the red ball as the answer. These frames will be converted into RelEx format by the RelEx2Frames rule based system as described above, and NLGen will generate the expected sentence "the red ball is next to the tree".

46.5.4 Example of the PetBrain Language Generation Pipeline

To illustrate the process of language generation using NLGen, as utilized in the context of PetBrain query response, consider the sentence "The red ball is near the tree". When parsed by RelEx, this sentence is converted to:

```
_obj(near, tree)
_subj(near, ball)
imperative(near)
hyp(near)
definite(tree)
singular(tree)
_to-do(be, near)
_subj(be, ball)
present(be)
definite(ball)
singular(ball)
```

So, if sentences with this format are in the system's experience, these relations are stored by NLGen and will be used to match future relations that must be converted into natural language. NLGen matches at an abstract level, so sentences like "The stick is next to the fountain" will also be matched even if the corpus contain only the sentence "The ball is near the tree".

If the agent wants to say that "The red ball is near the tree", it must invoke NLGen with the above RelEx contents as input. However, the knowledge that the red ball is near the tree is stored as frames, and not as RelEx format. More specifically, in this case the related frame stored is the *Locative_relation* one, containing the following elements and respective values: Figure → red ball, Ground → tree, *Relation_type* → near.

So we must convert these frames and their elements' values into the RelEx format accept by NLGen. For AGI purposes, a system must learn how to perform this conversion in a flexible and context-appropriate way. In our current system, however, we have implemented a temporary short-cut: a system of hand-coded rules, in which the preconditions are the required frames and the output is the corresponding RelEx format that will generate the sentence that represents the frames. The output of a rule may contains variables that must be replaced by the frame elements' values. For the example above, the output $\_subj(be, ball)$ is generated from the rule output $\_subj(be, \$var1)$ with the $\$var1$ replaced by the Figure element value.

Considering specifically question-answering (QA), the PetBrain's Language Comprehension module represents the answer to a question as a list of frames. In this case, we may have the following situations:

- The frames match a precondition and the RelEx output is correctly recognized by NLGen, which generates the expected sentence as the answer;
- The frames match a precondition, but NLGen did not recognize the RelEx output generated. In this case, the answer will be "I know the answer, but I don't know how to say it", which means that the question was answered correctly by the Language Comphrehension, but the NLGen could not generate the correct sentence;
- The frames didn't match any precondition; then the answer will also be "I know the answer, but I don't know how to say it" ;

- Finally, if no frames are generated as answer by the Language Comprehension module, the agent's answer will be "I don't know".

If the question is a truth-question, then NLGen is not required. In this case, the creation of frames as answer is considered as a "Yes", otherwise, the answer will be "No" because it was not possible to find the corresponding frames as the answer.

46.6 The Prospect of Massively Multiplayer Language Teaching

Now we tie in the theme of embodied language learning with more general considerations regarding embodied experiential learning.

Potentially, this may provide a means to facilitate robust language learning on the part of virtually embodied agents, and lead to an experientially-trained AGI language facility that can then be used to power other sorts of agents such as virtual babies, and ultimately virtual adult-human avatars that can communicate with experientially-grounded savvy rather than in the manner of chat-bots.

As one concrete, evocative example, imagine millions of talking parrots spread across different online virtual worlds – - all communicating in simple English. Each parrot has its own local memories, its own individual knowledge and habits and likes and dislikes – - but there's also a common knowledge-base underlying all the parrots, which includes a common knowledge of English.

The interest of many humans in interacting with chatbots suggests that virtual talking parrots or similar devices would be likely to meet with a large and enthusiastic audience.

Yes, humans interacting with parrots in virtual worlds can be expected to try to teach the parrots ridiculous things, obscene things, and so forth. But still, when it comes down to it, even pranksters and jokesters will have more fun with a parrot that can communicate better, and will prefer a parrot whose statements are comprehensible.

And for a virtual parrot, the test of whether it has used English correctly, in a given instance, will come down to whether its human friends have rewarded it, and whether it has gotten what it wanted. If a parrot asks for food incoherently, it's less likely to get food – - and since the virtual parrots will be programmed to want food, they will have motivation to learn to speak correctly. If a parrot interprets a human-controlled avatar's request "Fetch my hat please" incorrectly, then it won't get positive feedback from the avatar – - and it will be programmed to want positive feedback.

And of course parrots are not the end of the story. Once the collective wisdom of throngs of human teachers has induced powerful language understanding in the collective bird-brain, this language understanding (and the

commonsense understanding coming along with it) will be useful for many, many other purposes as well. Humanoid avatars – - both human-baby avatars that may serve as more rewarding virtual companions than parrots or other virtual animals; and language-savvy human-adult avatars serving various useful and entertaining functions in online virtual worlds and games. Once AIs have learned enough that they can flexibly and adaptively explore online virtual worlds and gather information from human-controlled avatars according to their own goals using their linguistic facilities, it's easy to envision dramatic acceleration in their growth and understanding.

A baby AI has numerous disadvantages compared to a baby human being: it lacks the intricate set of inductive biases built into the human brain, and it also lacks a set of teachers with a similar form and psyche to it... and for that matter, it lacks a really rich body and world. However, the presence of thousands to millions of teachers constitutes a large advantage for the AI over human babies. And a flexible AGI framework will be able to effectively exploit this advantage. If nonlinguistic learning mechanisms like the ones we've described here, utilized in a virtually-embodied context, can go beyond enabling interestingly trainable virtual animals and catalyze the process of language learning – then, within a few years time, we may find ourselves significantly further along the path to AGI than most observers of the field currently expect.

Chapter 47
Natural Language Dialogue

Co-authored with Ruiting Lian

47.1 Introduction

Language evolved for dialogue – not for reading, writing or speechifying. So it's natural that dialogue is broadly considered a critical aspect of humanlike AGI – even to the extent that (for better or for worse) the conversational "Turing Test" is the standard test of human-level AGI.

Dialogue is a high-level functionality rather than a foundational cognitive process, and in the CogPrime approach it is something that must largely be learned via experience rather than being programmed into the system. In that sense, it may seem odd to have a chapter on dialogue in a book section focused on *engineering* aspects of general intelligence. One might think: Dialogue is something that should emerge from an intelligent system in conjunction with other intelligent systems, not something that should need to be engineered. And this is certainly a reasonable perspective! We do think that, as a CogPrime system develops, it will develop its own approach to natural language dialogue, based on its own embodiment, environment and experience – with similarities and differences to human dialogue.

However, we have also found it interesting to design a natural language dialogue system based on CogPrime , with the goal **not** of emulating human conversation, but rather of enabling interesting and intelligent conversational interaction with CogPrime systems. We call this system "ChatPrime" and will describe its architecture in this chapter. The components used in ChatPrime may also be useful for enabling CogPrime systems to carry out more humanlike conversation, via their incorporation in learned schemata; but we will not focus on that aspect here. In addition to its intrinsic interest, consideration of ChatPrime sheds much light on the conceptual relationship between the NLP and other aspects of CogPrime .

We are aware that there is an active subfield of computational linguistics focused on dialogue systems [Wah06, **?**], however we will not draw significantly on that literature here. Making practical dialogue systems in the ab-

sence of a generally functional cognitive engine is a subtle and difficult art, which has been addressed in a variety of ways; however, we have found that designing a dialogue system within the context of an integrative cognitive engine like CogPrime is a somewhat different sort of endeavor.

47.1.1 Two Phases of Dialogue System Development

In practical terms, we envision the ChatPrime system as possessing two phases of development:

1. **Phase 1**:
   - "Lower levels" of NL comprehension and generation executed by a relatively traditional approach incorporating statistical and rule-based aspects (the RelEx and NLGen systems)
   - Dialogue control utilizes hand-coded procedures and predicates (SpeechActSchema and SpeechActTriggers) corresponding to fine-grained types of speech act
   - Dialogue control guided by general cognitive control system (OpenPsi, running within OpenCog)
   - SpeechActSchema and SpeechActTriggers, in some cases, will internally consult probabilistic inference, thus supplying a high degree of adaptive intelligence to the conversation
2. **Phase 2**:
   - "Lower levels" of NL comprehension and generation carried out within primary cognition engine, in a manner enabling their underlying rules and probabilities to be modified based the system's experience. Concretely, one way this could be done in OpenCog would be via
     - Implementing the RelEx and RelEx2Frame rules as PLN implications in the Atomspace
     - Implementing parsing via expressing the link parser dictionary as Atoms in the Atomspace, and using the SAT link parser to do parsing as an example of logical unification (carried out by a MindAgent wrapping an SAT solver)
     - Implementing NLGen within the OpenCog core, via making NLGen's sentence database a specially indexed Atomspace, and wrapping the NLGen operations in a MindAgent
   - Reimplement the SpeechActSchema and SpeechActTriggers in an appropriate combination of Combo and PLN logical link types, so they are susceptible to modification via inference and evolution

It's worth noting that the work required to move from Phase 1 to Phase 2 is essentially software development and computer science algorithm opti-

mization work, rather than computational linguistics or AI theory. Then after the Phase 2 system is built there will, of course, be significant work involved in "tuning" PLN, MOSES and other cognitive algorithms to experientially adapt the various portions of the dialogue system that have been moved into the OpenCog core and refactored for adaptiveness.

47.2 Speech Act Theory and its Elaboration

We review here the very basics of speech act theory, and then the specific variant of speech act theory that we feel will be most useful for practical OpenCog dialogue system development.

The core notion of speech act theory is to analyze linguistic behavior in terms of discrete speech acts aimed at achieving specific goals. This is a convenient theoretical approach in an OpenCog context, because it pushes us to treat speech acts just like any other acts that an OpenCog system may carry out in its world, and to handle speech acts via the standard OpenCog action selection mechanism.

Searle, who originated speech act theory, divided speech acts according to the following (by now well known) ontology:

- **Assertives** : The speaker commits herself to something being true. *The sky is blue.*
- **Directives**: The speaker attempts to get the hearer to do something. *Clean your room!*
- **Commissives**: The speaker commits to some future course of action. *I will do it.*
- **Expressives**: The speaker expresses some psychological state. *IÕm sorry.*
- **Declarations**: The speaker brings about a different state of the world. *The meeting is adjourned.*

Inspired by this ontology, Twitchell and Nunamaker (in their 2004 paper "Speech Act Profiling: A Probabilistic Method for Analyzing Persistent Conversations and Their Participants") created a much more fine-grained ontology of 42 kinds of speech acts, called SWBD-DAMSL (DAMSL = Dialogue Act Markup in Several Layers). Nearly all of their 42 speech act types can be neatly mapped into one of Searle's 5 high level categories, although a handful don't fit Searle's view and get categorized as "other." Figures 47.1 and 47.2 depict the 42 acts and their relationship to Searle's categories.

Tag Name	Tag	Example
STATEMENT-NON-OPINION	sd	Me, I'm in the legal department.
ACKNOWLEDGE (BACKCHANNEL)	b	Uh-huh.
STATEMENT-OPINION	sv	I think it's great.
AGREE/ACCEPT	aa	That's exactly it.
ABANDONED, TURN-EXIT OR UNINTERPRETABLE	%	So,-
APPRECIATION	ba	I can imagine.
YES-NO-QUESTION	qy	Do you have to have any special training?
NON-VERBAL	x	[Laughter], [Throat-clearing]
YES ANSWERS	ny	Yes.
CONVENTIONAL-CLOSING	fc	Well, it's been nice talking to you.
WH-QUESTION	qw	Well, how old are you?
NO ANSWERS	nn	No.
RESPONSE ACKNOWLEDGEMENT	bk	Oh, okay.
HEDGE	h	I don't know if I'm making any sense or not.
DECLARATIVE YES-NO-QUESTION	qyCd	So you can afford to get a house?
OTHER	other	Well give me a break, you know.
BACKCHANNEL IN QUESTION FORM	bh	Is that right?
QUOTATION	Cq	You can't be pregnant and have cats.
SUMMARIZE/REFORMULATE	bf	Oh, you mean you switched schools for the kids.
AFFIRMATIVE NON-YES ANSWERS	na	It is.
ACTION-DIRECTIVE	ad	Why don't you go first
COLLABORATIVE COMPLETION	C2	Who aren't contributing.
REPEAT-PHRASE	bCm	Oh, fajitas
OPEN-QUESTION	qo	How about you?
RHETORICAL-QUESTIONS	qh	Who would steal a newspaper?
HOLD BEFORE ANSWER/AGREEMENT	Ch	I'm drawing a blank.
REJECT	ar	Well, no
NEGATIVE NON-NO ANSWERS	ng	Uh, not a whole lot.
SIGNAL-NON-UNDERSTANDING	br	Excuse me?
OTHER ANSWERS	no	I don't know
CONVENTIONAL-OPENING	fp	How are you?
OR-CLAUSE	qrr	or is it more of a company?
DISPREFERRED ANSWERS	arp	Well, not so much that.
3RD-PARTY-TALK	t3	My goodness, Diane, get down from there.
OFFERS, OPTIONS COMMITS	commits	I'll have to check that out
SELF-TALK	t1	What's the word I'm looking for
DOWNPLAYER	bd	That's all right.
MAYBE/ACCEPT-PART	aap	Something like that
TAG-QUESTION	Cg	Right?
DECLARATIVE WH-QUESTION	qwCd	You are what kind of buff?
APOLOGY	fa	I'm sorry.
THANKING	ft	Hey thanks a lot

Fig. 47.1 The 42 DAMSL speech act categories.

Assertives	Expressives	Directives
STATEMENT	OPINION	YES-NO-QUESTION
YES ANSWERS	ABANDONED/UNINTERPRETABLE	WH-QUESTION
NO ANSWERS	BACKCHANNEL/ACKNOWLEDGE	DECLARATIVE YES-NO-QUESTION
QUOTATION	RESPONSE ACKNOWLEDGEMENT	BACKCHANNEL-QUESTION
AFFIRMATIVE NON-YES ANSWERS	SIGNAL-NON-UNDERSTANDING	SUMMARIZE/REFORMULATE
COLLABORATIVE COMPLETION	AGREEMENT/ACCEPT	ACTION-DIRECTIVE
RHETORICAL-QUESTIONS	APPRECIATION	OPEN-QUESTION
NEGATIVE NON-NO ANSWERS	CONVENTIONAL-CLOSING	TAG-QUESTION
OTHER ANSWERS	HEDGE	DECLARATIVE WH-QUESTION
OR-CLAUSE	HOLD BEFORE ANSWER/AGREEMENT	
DISPREFERRED ANSWERS	REJECT	Other
	CONVENTIONAL-OPENING	OTHER
Commissives	DOWNPLAYER	THIRD-PARTY TALK
OFFERS, OPTIONS, & COMMITS	MAYBE/ACCEPT-PART	NONVERBAL
	APOLOGY	
Declarations[1]	THANKING	
	REPEAT-PHRASE	

[1] None of the 42 dialogue acts could be described as a declaration

Fig. 47.2 Connecting the 42 DAMSL speech act categories to Searle's 5 higher-level categories.

47.3 Speech Act Schemata and Triggers

In the suggested dialogue system design, multiple SpeechActSchema would be implemented, corresponding *roughly* to the 42 SWBD-DAMSL speech acts. The correspondence is "rough" because

- we may wish to add new speech acts not in their list
- sometimes it may be most convenient to merge 2 or more of their speech acts into a single SpeechActSchema. For instance, it's probably easiest to merge their YES ANSWER and NO ANSWER categories into a single TRUTH VALUE ANSWER schema, yielding affirmative, negative, and intermediate answers like "probably", "probably not", "I'm not sure", etc.
- sometimes it may be best to split one of their speech acts into several, e.g. to separately consider STATEMENTs which are responses to statements, versus statements that are unsolicited disbursements of "what's on the agent's mind."

Overall, the SWBD-DAMSL categories should be taken as guidance rather than doctrine. However, they are valuable guidance due to their roots in detailed analysis of real human conversations, and their role as a bridge between concrete conversational analysis and the abstractions of speech act theory.

Each SpeechActSchema would take in an input consisting of a DialogueNode, a Node type possessing a collection of links to

- a series of past statements by the agent and other conversation participants, with
  - each statement labeled according to the utterer
  - each statement uttered by the agent, labeled according to which SpeechActSchema was used to produce it, plus (see below) which SpeechActTrigger and which response generator was involved
- a set of Atoms comprising the context of the dialogue. These Atoms may optionally be linked to some of the Atoms representing some of the past statements. If they are not so linked, they are considered as general context.

The enaction of SpeechActSchema would be carried out via PredictiveImplicationLinks embodying "Context AND Schema → Goal" schematic implications, of the general form

```
PredictiveImplication
   AND
        Evaluation
           SpeechActTrigger T
           DialogueNode D
        Execution
```

```
                SpeechActSchema S
                DialogueNode D
          Evaluation
               Evaluation
                  Goal G
```

with

```
ExecutionOutput
     SpeechActSchema S
     DialogueNode D
     UtteranceNode U
```

being created as a result of the enaction of the SpeechActSchema. (An UtteranceNode is a series of one or more SentenceNodes.)

A single SpeechActSchema may be involved in many such implications, with different probabilistic weights, if it naturally has many different Trigger contexts.

Internally each SpeechActSchema would contain a set of one or more response generators, each one of which is capable of independently producing a response based on the given input. These may also be weighted, where the weight determines the probability of a given response generation process being chosen in preference to the others, once the choice to enact that particular SpeechActSchema has already been made.

47.3.1 Notes Toward Example SpeechActSchema

To make the above ideas more concrete, let's consider a few specific SpeechActSchema. We won't fully specify them here, but will outline them sufficiently to make the ideas clear.

47.3.1.1 TruthValueAnswer

The TruthValueAnswer SpeechActSchema would encompass SWBD-DAMSL's YES ANSWER and NO ANSWER, and also more flexible truth value based responses.

Trigger context

: when the conversation partner produces an utterance that RelEx maps into a truth-value query (this is simple as truth-value-query is one of RelEx's relationship types).

Goal

: the simplest goal relevant here is pleasing the conversation partner, since the agent may have noticed in the past that other agents are pleased when their questions are answers. (More advanced agents may of course have other goals for answering questions, e.g. providing the other agent with information that will let it be more useful in future.)

Response generation schema

: for starters, this SpeechActSchema could simply operate as follows. It takes the relationship (Atom) corresponding to the query, and uses it to launch a query to the pattern matcher or PLN backward chainer. Then based on the result, it produces a relationship (Atom) embodying the answer to the query, or else updates the truth value of the existing relationship corresponding to the answer to the query. This "answer" relationship has a certain truth value. The schema could then contain a set of rules mapping the truth values into responses, with a list of possible responses for each truth value range. For example a very high strength and high confidence truth value would be mapped into a set of responses like {definitely, certainly, surely, yes, indeed}.

This simple case exemplifies the overall Phase 1 approach suggested here. The conversation will be guided by fairly simple heuristic rules, but with linguistic sophistication in the comprehension and generation aspects, and potentially subtle inference invoked within the SpeechActSchema or (less frequently) the Trigger contexts. Then in Phase 2 these simple heuristic rules will be refactored in a manner rendering them susceptible to experiential adaptation.

47.3.1.2 Statement: Answer

The next few SpeechActSchema (plus maybe some similar ones not given here) are intended to collectively cover the ground of SWBD-DAMSL's STATEMENT OPINION and STATEMENT NON-OPINION acts.

Trigger context

: The trigger is that the conversation partner asks a wh- question

Goal

: Similar to the case of a TruthValueAnswer, discussed above

Response generation schema

: When a wh- question is received, one reasonable response is to produce a statement comprising an answer. The question Atom is posed to the pattern matcher or PLN, which responds with an Atom-set comprising a putative answer. The answer Atoms are then pared down into a series of sentence-sized Atom-sets, which are articulated as sentences by NLGen. If the answer Atoms have very low-confidence truth values, or if the Atomspace contains knowledge that other agents significantly disagree with the agent's truth value assessments, then the answer Atom-set may have Atoms corresponding to "I think" or "In my opinion" etc. added onto it (this gives an instance of the STATEMENT NON-OPINION act).

47.3.1.3 Statement: Unsolicited Observation

Trigger context

: when in the presence of another intelligent agent (human or AI) and nothing has been said for a while, there is a certain probability of choosing to make a "random" statement.

Goal 1

: Unsolicited observations may be made with a goal of pleasing the other agent, as it may have been observed in the past that other agents are happier when spoken to

Goal 2

: Unsolicited observations may be made with goals of increasing the agent's own pleasure or novelty or knowledge – because it may have been observed that speaking often triggers conversations, and conversations are often more pleasurable or novel or educational than silence

Response generation schema

: One option is a statement describing something in the mutual environment, another option is a statement derived from high-STI Atoms in the agent's Atomspace. The particulars are similar to the "Statement: Answer" case.

47.3.1.4 Statement: External Change Notification

Trigger context

: when in a situation with another intelligent agent, and something significant changes in the mutually perceived situation, a statement describing it may be made.

Goal 1

: External change notification utterances may be made for the same reasons as Unsolicited Observations, described above.

Goal 2

: The agent may think a certain external change is important to the other agent it is talking to, for some particular reason. For instance, if the agent sees a dog steal Bob's property, it may wish to tell Bob about this.

Goal 3

: The change may be important to the agent itself – and it may want its conversation partner to do something relevant to an observed external change ... so it may bring the change to the partner's attention for this reason. For instance, "Our friends are leaving. Please try to make them come back."

Response generation schema

: The Atom-set for expression characterizes the change observed. The particulars are similar to the "Statement: Answer" case.

47.3.1.5 Statement: Internal Change Notification

Trigger context 1

: when the importance level of an Atom increases dramatically while in the presence of another intelligent agent, a statement expressing this Atom (and some of its currently relevant surrounding Atoms) may be made

Trigger context 2

: when the truth value of a reasonably important Atom changes dramatically while in the presence of another intelligent agent, a statement expressing this Atom and its truth value may be made

Goal

: Similar goals apply here as to External Change Notification, considered above

Response generation schema

: Similar to the "Statement: External Change Notification" case.

47.3.1.6 WHQuestion

Trigger context

: being in the presence of an intelligent agent thought capable of answering questions

Goal 1

: the general goal of increasing the agent's total knowledge

Goal 2

: the agent notes that, to achieve one of its currently important goals, it would be useful to possess a Atom fulfilling a certain specification

Response generation schema

: Formulate a query whose answer would be an Atom fulfilling that specification, and then articulate this logical query as an English question using NLGen

47.4 Probabilistic Mining of Trigger contexts

One question raised by the above design sketch is where the Trigger contexts come from. They may be hand-coded, but this approach may suffer from excessive brittleness. The approach suggested by Twitchell and Nunamaker's work (which involved modeling human dialogues rather than automatically generating intelligent dialogues) is statistical. That is, they suggest marking up a corpus of human dialogues with tags corresponding to the 42 speech acts, and learning from this annotated corpus a set of Markov transition probabilities indicating which speech acts are most likely to follow which others. In their approach the transition probabilities refer only to series of speech acts.

In an OpenCog context one could utilize a more sophisticated training corpus in a more sophisticated way. For instance, suppose one wants to build a dialogue system for a game character conversing with human characters in a game world. Then one could conduct experiments in which one human controls a "human" game character, and another human puppeteers an "AI" game character. That is, the puppeteered character funnels its perceptions to the AI system, but has its actions and verbalizations controlled by the human puppeteer. Given the dialogue from this sort of session, one could then perform markup according to the 42 speech acts.

As a simple example, consider the following brief snippet of annotated conversation:

speaker	**utterance**	**speech act type**
Ben	Go get me the ball	ad
AI	Where is it?	qw
Ben	Over there [points]	sd
AI	By the table?	qy
Ben	Yeah	ny
AI	Thanks	ft
AI	I'll get it now.	commits

A DialogueNode object based on this snippet would contain the information in the table, plus some physical information about the situation, such as, in this case: predicates describing the relative locations of the two agents, the ball an the table (e.g. the two agents are very near each other, the ball and

the table are very near each other, but these two groups of entities are only moderately near each other); and, predicates involving

Then, one could train a machine learning algorithm such as MOSES to predict the probability of speech act type S_1 occurring at a certain point in a dialogue history, based on the prior history of the dialogue. This prior history could include percepts and cognitions as well as utterances, since one has a record of the AI system's perceptions and cognitions in the course of the marked-up dialogue.

One question is whether to use the 42 SWBD-DAMSL speech acts for the creation of the annotated corpus, or whether instead to use the modified set of speech acts created in designing SpeechActSchema. Either way could work, but we are mildly biased toward the former, since this specific SWBD-DAMSL markup scheme has already proved its viability for marking up conversations. It seems unproblematic to map probabilities corresponding to these speech acts into probabilities corresponding to a slightly refined set of speech acts. Also, this way the corpus would be valuable independently of ongoing low-level changes in the collection of SpeechActSchema.

In addition to this sort of supervised training in advance, it will be important to enable the system to learn Trigger contexts online as a consequence of its life experience. This learning may take two forms:

1. Most simply, adjustment of the probabilities associated with the PredictiveImplicationLinks between SpeechActTriggers and SpeechActSchema
2. More sophisticatedly, learning of new SpeechActTrigger predicates, using an algorithms such as MOSES for predicate learning, based on mining the history of actual dialogues to estimate fitness

In both cases the basis for learning is information regarding the extent to which system goals were fulfilled by each past dialogue. PredictiveImplications that correspond to portions of successful dialogues will be have their truth values increased, and those corresponding to portions of unsuccessful dialogues will have their truth values decreased. Candidate SpeechActTriggers will be valued based on the observed historical success of the responses they would have generated based on historically perceived utterances; and (ultimately) more sophisticatedly, based on the estimated success of the responses they generate. Note that, while somewhat advanced, this kind of learning is much easier than th procedure learning required to learn new SpeechActSchema.

47.5 Conclusion

While the underlying methods are simple, the above methods appear capable of producing arbitrarily complex dialogues about any subject that is represented by knowledge in the AtomSpace. There is no reason why dialogue

produced in this manner should be indistinguishable from human dialogue; but it may nevertheless be humanly comprehensible, intelligent and insightful. What is happening in this sort of dialogue system is somewhat similar to current natural language query systems that query relational databases, but the "database" in question is a dynamically self-adapting weighted labeled hypergraph rather than a static relational database, and this difference means a much more complex dialogue system is required, as well as more flexible language comprehension and generation components.

Ultimately, a CogPrime system – if it works as desired – will be able to learn increased linguistic functionality, and new languages, on its own. But this is not a prerequisite for having intelligent dialogues with a CogPrime system. Via building a ChatPrime type system, as outlined here, intelligent dialogue can occur with a CogPrime system while it is still at relatively early stages of cognitive development, and even while the underlying implementation of the CogPrime design is incomplete. This is not closely analogous to human cognitive and linguistic development, but, it can still be pursued in the context of a CogPrime development plan that follows the overall arc of human developmental psychology.

Section XIII
From Here to AGI

Chapter 48
Summary of Argument for the CogPrime Approach

48.1 Introduction

By way of conclusion, we now return to the "key claims" that were listed at the end of Chapter 1 of Part 1. Quite simply, this is a list of claims such that – roughly speaking – if the reader accepts these claims, they should accept that the CogPrime approach to AGI is a viable one. On the other hand if the reader rejects one or more of these claims, they may well find one or more aspects of CogPrime unacceptable for some related reason. In Chapter 1 of Part 1 we merely listed these claims; here we briefly discuss each one in the context of the intervening chapters, giving each one its own section or subsection.

As we clarified at the start of Part 1, we don't fancy that we have provided an ironclad argument that the CogPrime approach to AGI is guaranteed to work as hoped, once it's fully engineered, tuned and taught. Mathematics isn't yet adequate to analyze the real-world behavior of complex systems like these; and we have not yet implemented, tested and taught enough of CogPrime to provide convincing empirical validation. So, most of the claims listed here have not been rigorously demonstrated, but only heuristically argued for. That is the reality of AGI work right now: one assembles a design based on the best combination of rigorous and heuristic arguments one can, then proceeds to create and teach a system according to the design, adjusting the details of the design based on experimental results as one goes along.

For an uncluttered list of the claims, please refer back to Chapter 1 of Part 1; here we will review the claims integrated into the course of discussion.

The following chapter, aimed at the more mathematically-minded reader, gives a list of formal propositions echoing many of the ideas in the chapter – propositions such that, if they are true, then the success of CogPrime as an architecture for general intelligence is likely.

48.2 Multi-Memory Systems

The first of our key claims is that *to achieve general intelligence in the context of human-intelligence-friendly environments and goals using feasible computational resources, it's important that an AGI system can handle different kinds of memory (declarative, procedural, episodic, sensory, intentional, attentional) in customized but interoperable ways*. The basic idea is that these different kinds of knowledge have very different characteristics, so that trying to handle them all within a single approach, while surely possible, is likely to be unacceptably inefficient.

The tricky issue in formalizing this claim is that "single approach" is an ambiguous notion: for instance, if one has a wholly logic-based system that represents all forms of knowledge using predicate logic, then one may still have specialized inference control heuristics corresponding to the different kinds of knowledge mentioned in the claim. In this case one has "customized but interoperable ways" of handling the different kinds of memory, and one doesn't really have a "single approach" even though one is using logic for everything. To bypass such conceptual difficulties, one may formalize cognitive synergy using a geometric framework as discussed in Appendix B, in which different types of knowledge are represented as metrized categories, and cognitive synergy becomes a statement about paths to goals being shorter in metric spaces combining multiple knowledge types than in those corresponding to individual knowledge types.

In CogPrime we use a complex combination of representations, including the Atomspace for declarative, attentional and intentional knowledge and some episodic and sensorimotor knowledge, Combo programs for procedural knowledge, simulations for episodic knowledge, and hierarchical neural nets for some sensorimotor knowledge (and related episodic, attentional and intentional knowledge). In cases where the same representational mechanism is used for different types of knowledge, different cognitive processes are used, and often different aspects of the representation (e.g. attentional knowledge is dealt with largely by ECAN acting on AttentionValues and HebbianLinks in the Atomspace; whereas declarative knowledge is dealt with largely by PLN acting on TruthValues and logical links, also in the AtomSpace). So one has a mix of the "different representations for different memory types" approach and the "different control processes on a common representation for different memory types" approach.

It's unclear how closely dependent the need for a multi-memory approach is on the particulars of "human-friendly environments." We argued in Chapter 9 of Part 1 that one factor militating in favor of a multi-memory approach is the need for multimodal communication: declarative knowledge relates to linguistic communication; procedural knowledge relates to demonstrative communication; attentional knowledge relates to indicative communication; and so forth. But in fact the multi-memory approach may have a broader importance, even to intelligences without multimodal communication. This is an

interesting issue but not particularly critical to the development of human-like, human-level AGI, since in the latter case we are specifically concerned with creating intelligences that *can* handle multimodal communication. So if for no other reason, the multi-memory approach is worthwhile for handling multi-modal communication.

Pragmatically, it is also quite clear that the human brain takes a multi-memory approach, e.g. with the cerebellum and closely linked cortical regions containing special structures for handling procedural knowledge, with special structures for handling motivational (intentional) factors, etc. And (though this point is certainly not definitive, it's meaningful in the light of the above theoretical discussion) decades of computer science and narrow-AI practice strongly suggest that the "one memory structure fits all" approach is not capable of leading to effective real-world approaches.

48.3 Perception, Action and Environment

The more we understand of human intelligence, the clearer it becomes how closely it has evolved to the particular goals and environments for which the human organism evolved. This is true in a broad sense, as illustrated by the above issues regarding multi-memory systems, and is also true in many particulars, as illustrated e.g. by Changizi's [Cha09] evolutionary analysis of the human visual system. While it might be possible to create a human-like, human-level AGI by abstracting the relevant biases from human biology and behavior and explicitly encoding them in one's AGI architecture, it seems this would be an inordinately difficult approach in practice, leading to the claim that *to achieve human-like general intelligence, it's important for an intelligent agent to have sensory data and motoric affordances that roughly emulate those available to humans*. We don't claim this is a necessity – just a dramatic convenience. And if one accepts this point, it has major implications for what sorts of paths toward AGI it makes most sense to follow.

Unfortunately, though, the idea of a "human-like" set of goals and environments is fairly vague; and when you come right down to it, *we don't know exactly how close the emulation needs to be* to form a natural scenario for the maturation of human-like, human-level AGI systems. One could attempt to resolve this issue via a priori theory, but given the current level of scientific knowledge it's hard to see how that would be possible in any definitive sense ... which leads to the conclusion that *our AGI systems and platforms need to support fairly flexible experimentation with virtual-world and/or robotic infrastructures*.

Our own intuition is that currently neither current virtual world platforms, nor current robotic platforms, are *quite* adequate for the development of human-level, human-like AGI. Virtual worlds would need to become a lot more like robot simulators, allowing more flexible interaction with the en-

vironment, and more detailed control of the agent. Robots would need to become more robust at moving and grabbing – e.g. with Big Dog's movement ability but the grasping capability of the best current grabber arms. We do feel that development of adequate virtual world or robotics platforms is quite possible using current technology, and could be done at fairly low cost if someone were to prioritize this. Even without AGI-focused prioritization, it seems that the needed technological improvements are likely to happen during the next decade for other reasons. So at this point we feel it makes sense for AGI researchers to focus on AGI and exploit embodiment-platform improvements as they come along – at least, this makes sense in the case of AGI approaches (like CogPrime) that can be primarily developed in an embodiment-platform-independent manner.

48.4 Developmental Pathways

But if an AGI system is going to live in human-friendly environments, what should it do there? No doubt very many pathways leading from incompetence to adult-human-level general intelligence exist, but one of them is much better understood than any of the others, and that's the one normal human children take. Of course, given their somewhat different embodiment, it doesn't make sense to try to force AGI systems to take *exactly* the same path as human children, but having AGI systems follow a fairly close approximation to the human developmental path seems the smoothest developmental course ... a point summarized by the claim that: *To work toward adult human-level, roughly human-like general intelligence, one fairly easily comprehensible path is to use environments and goals reminiscent of human childhood, and seek to advance one's AGI system along a path roughly comparable to that followed by human children.*

Human children learn via a rich variety of mechanisms; but broadly speaking one conclusion one may drawn from studying human child learning is that it may make sense to *teach an AGI system aimed at roughly human-like general intelligence via a mix of spontaneous learning and explicit instruction, and to instruct it via a combination of imitation, reinforcement and correction, and a combination of linguistic and nonlinguistic instruction.* We have explored exactly what this means in Chapter 31 and others, via looking at examples of these types of learning in the context of virtual pets in virtual worlds, and exploring how specific CogPrime learning mechanisms can be used to achieve simple examples of these types of learning.

One important case of learning that human children are particularly good at is language learning; and we have argued that this is a case where it may pay for AGI systems to take a route somewhat different from the one taken by human children. Humans seem to be born with a complex system of biases enabling effective language learning, and it's not yet clear exactly what

these biases are nor how they're incorporated into the learning process. It is very tempting to give AGI systems a "short cut" to language proficiency via making use of existing rule-based and statistical-corpus-analysis-based NLP systems; and we have fleshed out this approach sufficiently to have convinced ourselves it makes practical as well as conceptual sense, in the context of the specific learning mechanisms and NLP tools built into OpenCog. Thus we have provided a number of detailed arguments and suggestions in support of our claim that *one effective approach to teaching an AGI system human language is to supply it with some in-built linguistic facility, in the form of rule-based and statistical-linguistics-based NLP systems, and then allow it to improve and revise this facility based on experience.*

48.5 Knowledge Representation

Many knowledge representation approaches have been explored in the AI literature, and ultimately many of these could be workable for human-level AGI if coupled with the right cognitive processes. The key goal for a knowledge representation for AGI should be *naturalness* with respect to the AGI's cognitive processes – i.e. the cognitive processes shouldn't need to undergo complex transformative gymnastics to get information in and out of the knowledge representation in order to do their cognitive work. Toward this end we have come to a similar conclusion to some other researchers (e.g. Joscha Bach and Stan Franklin), and concluded that *given the strengths and weaknesses of current and near-future digital computers, a (loosely) neural-symbolic network is a good representation for directly storing many kinds of memory, and interfacing between those that it doesn't store directly.* CogPrime 's AtomSpace is a neural-symbolic network designed to work nicely with PLN, MOSES, ECAN and the other key CogPrime cognitive processes; it supplies them with what they need without causing them undue complexities. It provides a platform that these cognitive processes can use to adaptively, automatically construct specialized knowledge representations for particular sorts of knowledge that they encounter.

48.6 Cognitive Processes

The crux of intelligence is dynamics, learning, adaptation; and so the crux of an AGI design is the set of cognitive processes that the design provides. These processes must collectively allow the AGI system to achieve its goals in its environments using the resources at hand. Given CogPrime 's multi-memory design, it's natural to consider CogPrime 's cognitive processes in terms of which memory subsystems they focus on (although, this is not a

perfect mode of analysis, since some of the cognitive processes span multiple memory types).

48.6.1 Uncertain Logic for Declarative Knowledge

One major decision made in the creation of CogPrime was that *given the strengths and weaknesses of current and near-future digital computers, uncertain logic is a good way to handle declarative knowledge.* Of course this is not obvious nor is it the only possible route. Declarative knowledge can potentially be handled in other ways; e.g. in a hierarchical network architecture, one can make declarative knowledge emerge automatically from procedural and sensorimotor knowledge, as is the goal in the Numenta and DeSTIN designs reviewed in Chapter 4 of Part 1. It seems clear that the human brain doesn't contain anything closely parallel to formal logic – even though one can ground logic operations in neural-net dynamics as explored in Chapter 34, this sort of grounding leads to "uncertain logic enmeshed with a host of other cognitive dynamics" rather than "uncertain logic as a cleanly separable cognitive process."

But contemporary digital computers are not brains – they lack the human brain's capacity for cheap massive parallelism, but have a capability for single-operation speed and precision far exceeding the brain's. In this way computers and formal logic are a natural match (a fact that's not surprising given that Boolean logic lies at the foundation of digital computer operations). Using *uncertain logic* is a sort of compromise between brainlike messiness and fuzziness, and computerlike precision. An alternative to using uncertain logic is using crisp logic and incorporating uncertainty as content within the knowledge base – this is what SOAR does, for example, and it's not a wholly unworkable approach. But given that the vast mass of knowledge needed for confronting everyday human reality is highly uncertain, and that this knowledge often needs to be manipulated efficiently in real-time, it seems to us there is a strong argument for embedding uncertainty in the logic.

Many approaches to uncertain logic exist in the literature, including probabilistic and fuzzy approaches, and one conclusion we reached in formulating CogPrime is that none of them was adequate on its own – leading us, for example, to the conclusion that *to deal with the problems facing a human-level AGI, an uncertain logic must integrate imprecise probability and fuzziness with a broad scope of logical constructs.* The arguments that both fuzziness and probability are needed seem hard to counter – these two notions of uncertainty are qualitatively different yet both appear cognitively necessary.

The argument for using probability in an AGI system is assailed by some AGI researchers such as Pei Wang, but we are swayed by the theoretical arguments in favor of probability theory's mathematically fundamental nature,

as well as the massive demonstrated success of probability theory in various areas of narrow AI and applied science. However, we are also swayed by the arguments of Pei Wang, Peter Walley and others that using single-number probabilities to represent truth values leads to untoward complexities related to the tabulation and manipulation of amounts of evidence. This has led us to an imprecise probability based approach; and then technical arguments regarding the limitations of standard imprecise probability formalisms has led us to develop our own "indefinite probabilities" formalism.

The PLN logic framework is one way of integrating imprecise probability and fuzziness in a logical formalism that encompasses a broad scope of logical constructs. It integrates term logic and predicate logic – a feature that we consider not necessary, but very convenient, for AGI. Either predicate or term logic on its own would suffice, but each is awkward in certain cases, and integrating them as done in PLN seems to result in more elegant handling of real-world inference scenarios. Finally, PLN also integrates intensional inference in an elegant manner that demonstrates integrative intelligence – it defines intension using *pattern theory*, which binds inference to pattern recognition and hence to other cognitive processes in a conceptually appropriate way.

Clearly PLN is not the only possible logical formalism capable of serving a human-level AGI system; however, we know of no other existing, fleshed-out formalism capable of fitting the bill. In part this is because PLN has been developed as part of an integrative AGI project whereas other logical formalisms have mainly been developed for other purposes, or purely theoretically. Via using PLN to control virtual agents, and integrating PLN with other cognitive processes, we have tweaked and expanded the PLN formalism to serve all the roles required of the "declarative cognition" component of an AGI system with reasonable elegance and effectiveness.

48.6.2 Program Learning for Procedural Knowledge

Even more so than declarative knowledge, procedural knowledge is represented in many different ways in the AI literature. The human brain also apparently uses multiple mechanisms to embody different kinds of procedures. So the choice of how to represent procedures in an AGI system is not particularly obvious. However, there is one particular representation of procedures that is particularly well-suited for current computer systems, and particularly well-tested in this context: *programs*. In designing CogPrime , we have acted based on the understanding that *programs are a good way to represent procedures – including both cognitive and physical-action procedures, but perhaps not including low-level motor-control procedures.*

Of course, this begs the question of *programs in what programming language*, and in this context we have made a fairly traditional choice, using a

special language called Combo that is essentially a minor variant of LISP, and supplying Combo with a set of customized primitives intended to reduce the length of the typical programs CogPrime needs to learn and use. What differentiates this use of LISP from many traditional uses of LISP in AI is that we are *only* using the LISP-ish representational style for procedural knowledge, rather than trying to use it for everything.

One test of whether the use of Combo programs to represent procedural knowledge makes sense is whether the procedures useful for a CogPrime system in everyday human environments have short Combo representations. We have worked with Combo enough to validate that they generally do in the virtual world environment – and also in the physical-world environment *if* lower-level motor procedures are supplied as primitives. That is, we are not convinced that Combo is a good representation for the procedure a robot needs to do to move its fingers to pick up a cup, coordinating its movements with its visual perceptions. It's certainly possible to represent this sort of thing in Combo, but Combo may be an awkward tool. However, if one represents low-level procedures like this using another method, e.g. learned cell assemblies in a hierarchical network like DeSTIN, then it's very feasible to make Combo programs that invoke these low-level procedures, and encode higher-level actions like "pick up the cup in front of you slowly and quietly, then hand it to Jim who is standing next to you."

Having committed to use programs to represent many procedures, the next question is how to learn programs. One key conclusion we have come to via our empirical work in this area is that some form of powerful *program normalization* is essential. Without normalization, it's too hard for existing learning algorithms to generalize from known, tested programs and draw useful uncertain conclusions about untested ones. We have worked extensively with a generalization of Holman's "Elegant Normal Form" in this regard.

For learning normalized programs, we have come to the following conclusions:

- for relatively straightforward procedure learning problems, *hillclimbing with random restart and a strong Occam bias is an effective method*
- for more difficult problems that elude hillclimbing, *probabilistic evolutionary program learning is an effective method*

The probabilistic evolutionary program learning method we have worked with most in OpenCog is MOSES, and significant evidence has been gathered showing it to be dramatically more effective than genetic programming on relevant classes of problems. However, more work needs to be done to evaluate its progress on complex and difficult procedure learning problems. Alternate, related probabilistic evolutionary program learning algorithms such as PLEASURE have also been considered and may be implemented and tested as well.

48.6.3 Attention Allocation

There is significant evidence that the brain uses some sort of "activation spreading" type method to allocate attention, and many algorithms in this spirit have been implemented and utilized in the AI literature. So, we find ourselves in agreement with many others that *activation spreading is a reasonable way to handle attentional knowledge* (though other approaches, with greater overhead cost, may provide better accuracy and may be appropriate in some situations). We also agree with many others who have chosen **Hebbian learning as one route of learning associative relationships**, with more sophisticated methods such as information-geometric ones potentially also playing a role

Where CogPrime differs from standard practice is in the use of an economic metaphor to regulate activation spreading. In this matter CogPrime is broadly in agreement with Eric Baum's arguments about the value of economic methods in AI, although our specific use of economic methods is very different from his. Baum's work (e.g. Hayek) embodies more complex and computationally expensive uses of artificial economics, whereas we believe that *in the context of a neural-symbolic network, artificial economics is an effective approach to* **activation spreading**; and CogPrime 's ECAN framework seeks to embody this idea. ECAN can also make use of more sophisticated and expensive uses of artificial currency when large amount of system resources are involved in a single choice, rendering the cost appropriate.

One major choice made in the CogPrime design is *to focus on two kinds of attention: processor (represented by ShortTermImportance) and memory (represented by LongTermImportance)*. This is a direct reflection of one of the key differences between the von Neumann architecture and the human brain: in the former but not the latter, there is a strict separation between memory and processing in the underlying compute fabric. We carefully considered the possibility of using a larger variety of attention values, and in Chapter 23 we presented some mathematics and concepts that could be used in this regard, but for reasons of simplicity and computational efficiency we are currently using only STI and LTI in our OpenCogPrime implementation, with the possibility of extending further if experimentation proves it necessary.

48.6.4 Internal Simulation and Episodic Knowledge

For episodic knowledge, as with declarative and procedural knowledge, CogPrime has opted for a solution motivated by the particular strengths of contemporary digital computers. When the human brain runs through a "mental movie" of past experiences, it doesn't do any kind of accurate physical simulation of these experiences. But that's not because the brain wouldn't benefit from such – it's because the brain doesn't know how to do that sort of thing!

On the other hand, any modern laptop can run a reasonable Newtonian physics simulation of everyday events, and more fundamentally can recall and manage the relative positions and movements of items in an internal 3D landscape paralleling remembered or imagined real-world events. With this in mind, we believe that in an AGI context, *simulation is a good way to handle episodic knowledge; and running an internal "world simulation engine" is an effective way to handle simulation.*

CogPrime can work with many different simulation engines; and since simulation technology is continually advancing independently of AGI technology, this is an area where AGI can buy some progressive advancement for free as time goes on. The subtle issues here regard interfacing between the simulation engine and the rest of the mind: mining meaningful information out of simulations using pattern mining algorithms; and more subtly, figuring out what simulations to run at what times in order to answer the questions most relevant to the AGI system in the context of achieving its goals. We believe we have architected these interactions in a viable way in the CogPrime design, but we have tested our ideas in this regard only in some fairly simple contexts regarding virtual pets in a virtual world, and much more remains to be done here.

48.6.5 Low-Level Perception and Action

The centrality or otherwise of low-level perception and action in human intelligence is a matter of ongoing debate in the AI community. Some feel that the essence of intelligence lies in cognition and/or language, with perception and action having the status of "peripheral devices." Others feel that modeling the physical world and one's actions in it is the essence of intelligence, with cognition and language emerging as side-effects of these more fundamental capabilities. The CogPrime architecture doesn't need to take sides in this debate. Currently we are experimenting both in virtual worlds, and with real-world robot control. The value added by robotic versus virtual embodiment can thus be explored via experiment rather than theory, and may reveal nuances that no one currently foresees.

As noted above, we are unconfident of CogPrime 's generic procedure learning or pattern recognition algorithms in terms of their capabilities to handle large amounts of raw sensorimotor data in real time, and so for robotic applications we advocate hybridizing CogPrime with a separate (but closely cross-linked) system better customized for this sort of data, in line with our general hypothesis that *Hybridization of one's integrative neural-symbolic system with a spatiotemporally hierarchical deep learning system is an effective way to handle representation and learning of low-level sensorimotor knowledge.* While this general principle doesn't depend on any particular ap-

proach, DeSTIN is one example of a deep learning system of this nature that can be effective in this context

We have not yet done any sophisticated experiments in this regard – our current experiments using OpenCog to control robots involve cruder integration of OpenCog with perceptual and motor subsystems, rather than the tight hybridization described in Chapter 26. Creating such a hybrid system is "just" a matter of software engineering, but testing such a system may lead to many surprises!

48.6.6 Goals

Given that we have characterized general intelligence as "the ability to achieve complex goals in complex environments," it should be plain that goals play a central role in our work. However, we have chosen not to create a separate subsystem for intentional knowledge, and instead have concluded that *one effective way to handle goals is to represent them declaratively, and allocate attention among them economically.* An advantage of this approach is that it automatically provides integration between the goal system and the declarative and attentional knowledge systems.

Goals and subgoals are related using logical links as interpreted and manipulated by PLN, and attention is allocated among goals using the STI dynamics of ECAN, and a specialized variant based on RFS's (requests for service). Thus the mechanics of goal management is handled using uncertain inference and artificial economics, whereas the figuring-out of how to achieve goals is done integratively, relying heavily on procedural and episodic knowledge as well as PLN and ECAN.

The combination of ECAN and PLN seems to overcome the well-known shortcomings found with purely neural-net or purely inferential approaches to goals. Neural net approaches generally have trouble with abstraction, whereas logical approaches are generally poor at real-time responsiveness and at tuning their details quantitatively based on experience. At least in principle, our hybrid approach overcomes all these shortcomings; though of current, it has been tested only in fairly simple cases in the virtual world.

48.7 Fulfilling the "Cognitive Equation"

A key claim based on the notion of the "Cognitive Equation" posited in *Chaotic Logic* [Goe94] is that *it is important for an intelligent system to have some way of recognizing large-scale patterns in itself, and then embodying these patterns as new, localized knowledge items in its memory.* This dynamic introduces a feedback dynamic between emergent pattern and sub-

strate, which is hypothesized to be critical to general intelligence under feasible computational resources. It also ties in nicely with the notion of "glocal memory" – essentially positing a localization of some global memories, which naturally will result in the formation of some glocal memories. One of the key ideas underlying the CogPrime design is that *given the use of a neural-symbolic network for knowledge representation, a graph-mining based "map formation" heuristic is one good way to do this.*

Map formation seeks to fulfill the Cognitive Equation quite directly, probably more directly than happens in the brain. Rather than relying on other cognitive processes to implicitly recognize overall system patterns and embody them in the system as localized memories (though this implicit recognition may also happen), the MapFormation MindAgent explicitly carries out this process. Mostly this is done using fairly crude greedy pattern mining heuristics, though if really subtle and important patterns seem to be there, more sophisticated methods like evolutionary pattern mining may also be invoked.

It seems possible that this sort of explicit approach could be less efficient than purely implicit approaches; but, there is no evidence for this, and it may also provided *increased* efficiency. And in the context of the overall CogPrime design, the explicit MapFormation approach seems most natural.

48.8 Occam's Razor

The key role of "Occam's Razor" or the urge for simplicity in intelligence has been observed by many before (going back at least to Occam himself, and probably earlier!), and is fully embraced in the CogPrime design. Our theoretical analysis of intelligence, presented in Chapter 2 of Part 1 and elsewhere, portrays intelligence as closely tied to the creation of procedures that achieve goals in environments *in the simplest possible way.* And this quest for simplicity is present in many places throughout the CogPrime design, for instance

- In MOSES and hillclimbing, where program compactness is an explicit component of program tree fitness
- In PLN, where the backward and forward chainers explicitly favor shorter proof chains, and intensional inference explicitly characterizes entities in terms of their patterns (where patterns are defined as compact characterizations)
- In pattern mining heuristics, which search for compact characterizations of data
- In the forgetting mechanism, which seeks the smallest set of Atoms that will allow the regeneration of a larger set of useful Atoms via modestly-expensive application of cognitive processes

- Via the encapsulation of procedural and declarative knowledge in simulations, which in many cases provide a vastly compacted form of storing real-world experiences

Like cognitive synergy and emergent networks, Occam's Razor is not something that is implemented in a single place in the CogPrime design, but rather an overall design principle that underlies nearly every part of the system.

48.8.1 Mind Geometry

What about the three mind-geometric principles outlined in Appendix B:

- syntax-semantics correlation
- cognitive geometrodynamics
- cognitive synergy

?

The key role of syntax-semantics correlation in CogPrime is clear. It plays an explicit role in MOSES. In PLN, it is critical to inference control, to the extent that inference control is based on the extraction of patterns from previous inferences. The syntactic structures are the inference trees, and the semantic structures are the inferential conclusions produced by the trees. History-guided inference control assumes that prior similar trees will be a good starting-point for getting results similar to prior ones – i.e. it assumes a reasonable degree of syntax-semantics correlation. Also, without a correlation between the core elements used to generate an episode, and the whole episode, it would be infeasible to use historical data mining to understand what core elements to use to generate a new episode – and creation of compact, easily manipulable seeds for generating episodes would not be feasible.

Cognitive geometrodynamics is about finding the shortest path from the current state to a goal state, where distance is judged by an appropriate metric including various aspects of computational effort. The ECAN and effort management frameworks attempt to enforce this, via minimizing the amount of effort spent by the system in getting to a certain conclusion. MindAgents operating primarily on one kind of knowledge (e.g. MOSES, PLN) may for a time seek to follow the shortest paths within their particular corresponding memory spaces; but then when they operate more interactively and synergetically, it becomes a matter of finding short paths in the composite mindspace corresponding to the combination of the various memory types.

Finally, cognitive synergy is thoroughly and subtly interwoven throughout CogPrime . In a way the whole design is about cognitive synergy – it's critical for the design's functionality that *it's important that the cognitive processes associated with different kinds of memory can appeal to each other for assistance in overcoming bottlenecks in a manner that: a) works in "real time", i.e.*

on the time scale of the cognitive processes internal processes; b) enables each cognitive process to act in a manner that is sensitive to the particularities of each others' internal representations.

Recapitulating in a bit more depth, recall that another useful way to formulate cognitive synergy as follows. Each of the key learning mechanisms underlying CogPrime is susceptible to combinatorial explosions. As the problems they confront become larger and larger, the performance gets worse and worse at an exponential rate, because the number of combinations of items that must be considered to solve the problems grows exponentially with the problem size. This could be viewed as a deficiency of the fundamental design, but we don't view it that way. Our view is that combinatorial explosion is intrinsic to intelligence. The task at hand is to dampen it sufficiently that realistically large problems can be solved, rather than to eliminate it entirely. One possible way to dampen it would be to design a single, really clever learning algorithm - one that was still susceptible to an exponential increase in computational requirements as problem size increases, but with a surprisingly small exponent. Another approach is the mirrorhouse approach: Design a bunch of learning algorithms, each focusing on different aspects of the learning process, and design them so that they each help to dampen each others' combinatorial explosions. This is the approach taken within CogPrime . The component algorithms are clever on their own - they are less susceptible to combinatorial explosion than many competing approaches in the narrow-AI literature. But the real meat of the design lies in the intended interactions between the components, manifesting cognitive synergy.

48.9 Cognitive Synergy

To understand more specifically how cognitive synergy works in CogPrime , in the following subsections we will review some synergies related to the key components of CogPrime as discussed above. These synergies are absolutely critical to the proposed functionality of the CogPrime system. Without them, the cognitive mechanisms are not going to work adequately well, but are rather going to succumb to combinatorial explosions. The other aspects of CogPrime - the cognitive architecture, the knowledge representation, the embodiment framework and associated developmental teaching methodology - are all critical as well, but none of these will yield the critical emergence of intelligence without cognitive mechanisms that effectively scale. And, in the absence of cognitive mechanisms that effectively scale *on their own*, we must rely on cognitive mechanisms that *effectively help each other to scale*. The reasons why we believe these synergies will exist are essentially qualitative: we have not proved theorems regarded these synergies, and we have observed them in practice only in simple cases so far. However, we do have some ideas

regarding how to potentially prove theorems related to these synergies, and some of these are described in Appendix H.

48.9.1 Synergies that Help Inference

The combinatorial explosion in PLN is obvious: forward and backward chaining inference are both fundamentally explosive processes, reined in only by pruning heuristics. This means that for nontrivial complex inferences to occur, one needs **really, really clever** pruning heuristics. The CogPrime design combines simple heuristics with pattern mining, MOSES and economic attention allocation as pruning heuristics. Economic attention allocation assigns importance levels to Atoms, which helps guide pruning. Greedy pattern mining is used to search for patterns in the stored corpus of inference trees, to see if there are any that can be used as analogies for the current inference. And MOSES comes in when there is not enough information (from importance levels or prior inference history) to make a choice, yet exploring a wide variety of available options is unrealistic. In this case, MOSES tasks may be launched, pertinently to the leaves at the fringe of the inference tree, under consideration for expansion. For instance, suppose there is an Atom A at the fringe of the inference tree, and its importance hasn't been assessed with high confidence, but a number of items B are known so that:

```
MemberLink A B
```

Then, MOSES may be used to learn various relationships characterizing A, based on recognizing patterns across the set of B that are suspected to be members of A. These relationships may then be used to assess the importance of A more confidently, or perhaps to enable the inference tree to match one of the patterns identified by pattern mining on the inference tree corpus. For example, if MOSES figures out that:

```
SimilarityLink G A
```

then it may happen that substituting G in place of A in the inference tree, results in something that pattern mining can identify as being a good (or poor) direction for inference.

48.10 Synergies that Help MOSES

MOSES's combinatorial explosion is obvious: the number of possible programs of size N increases very rapidly with N. The only way to get around this is to utilize prior knowledge, and as much as possible of it. When solving a particular problem, the search for new solutions must make use of prior

candidate solutions evaluated for that problem, and also prior candidate solutions (including successful and unsuccessful ones) evaluated for other related problems.

But, extrapolation of this kind is in essence a contextual analogical inference problem. In some cases it can be solved via fairly straightforward pattern mining; but in subtler cases it will require inference of the type provided by PLN. Also, attention allocation plays a role in figuring out, for a given problem A, which problems B are likely to have the property that candidate solutions for B are useful information when looking for better solutions for A.

48.10.1 Synergies that Help Attention Allocation

Economic attention allocation, without help from other cognitive processes, is just a very simple process analogous to "activation spreading" and "Hebbian learning" in a neural network. The other cognitive processes are the things that allow it to more sensitively understand the attentional relationships between different knowledge items (e.g. which sorts of items are often usefully thought about in the same context, and in which order).

48.10.2 Further Synergies Related to Pattern Mining

Statistical, greedy pattern mining is a simple process, but it nevertheless can be biased in various ways by other, more subtle processes.

For instance, if one has learned a population of programs via MOSES, addressing some particular fitness function, then one can study which items tend to be utilized in the same programs in this population. One may then direct pattern mining to find patterns combining these items found to be in the MOSES population. And conversely, relationships denoted by pattern mining may be used to probabilistically bias the models used within MOSES.

Statistical pattern mining may also help PLN by supplying it with information to work on. For instance, conjunctive pattern mining finds conjunctions of items, which may then be combined with each other using PLN, leading to the formation of more complex predicates. These conjunctions may also be fed to MOSES as part of an initial population for solving a relevant problem.

Finally, the main interaction between pattern mining and MOSES/PLN is that the former may recognize patterns in links created by the latter. These patterns may then be fed back into MOSES and PLN as data. This virtuous cycle allows pattern mining and the other, more expensive cognitive processes to guide each other. Attention allocation also gets into the game,

by guiding statistical pattern mining and telling it which terms (and which combinations) to spend more time on.

48.10.3 Synergies Related to Map Formation

The essential synergy regarding map formation is obvious: Maps are formed based on the HebbianLinks created via PLN and simpler attentional dynamics, which are based on which Atoms are usefully used together, which is based on the dynamics of the cognitive processes doing the "using." On the other hand, once maps are formed and encapsulated, they feed into these other cognitive processes. This synergy in particular is critical to the emergence of self and attention.

What has to happen, for map formation to work well, is that the cognitive processes must *utilize* encapsulated maps in a way that gives rise overall to relatively clear clusters in the network of HebbianLinks. This will happen if the encapsulated maps are not too complex for the system's other learning operations to understand. So, there must be useful coordinated attentional patterns whose corresponding encapsulated-map Atoms are not too complicated. This has to do with the system's overall parameter settings, but largely with the settings of the attention allocation component. For instance, this is closely tied in with the limited size of "attentional focus" (the famous 7 +/- 2 number associated with humans' and other mammals short term memory capacity). If only a small number of Atoms are typically very important at a given point in time, then the maps formed by grouping together all simultaneously highly important things will be relatively small predicates, which will be easily reasoned about - thus keeping the "virtuous cycle" of map formation and comprehension going effectively.

48.11 Emergent Structures and Dynamics

We have spent much more time in this book on the engineering of cognitive processes and structures, than on the cognitive processes and structures that must *emerge* in an intelligent system for it to display human-level AGI. However, this focus should not be taken to represent a lack of appreciation for the importance of emergence. Rather, it represents a practical focus: engineering is what we must do to create a software system potentially capable of AGI, and emergence is then what happens inside the engineered AGI to allow it to achieve intelligence. Emergence must however be taken carefully into account when deciding what to engineer!

One of the guiding ideas underlying the CogPrime design is that *an AGI system with adequate mechanisms for handling the key types of knowledge*

mentioned above, and the capability to explicitly recognize large-scale pattern in itself, should **upon sustained interaction with an appropriate environment in pursuit of appropriate goals***, emerge a variety of complex structures in its internal knowledge network, including (but not limited to): a hierarchical network, representing both a spatiotemporal hierarchy and an approximate "default inheritance" hierarchy, cross-linked; a heterarchical network of associativity, roughly aligned with the hierarchical network; a self network which is an approximate micro image of the whole network; and inter-reflecting networks modeling self and others, reflecting a "mirrorhouse" design pattern.*

The dependence of these posited emergences on the environment and goals of the AGI system should not be underestimated. For instance, PLN and pattern mining don't have to lead to a hierarchical structured Atomspace, but if the AGI system is placed in an environment which it itself hierarchically structured, then they very likely will do so. And if this environment consists of hierarchically structured *language and culture*, then what one has is a system of minds with hierarchical networks, each reinforcing the hierarchality of each others' networks. Similarly, integrated cognition doesn't have to lead to mirrorhouse structures, but integrated cognition about situations involving other minds studying and predicting and judging each other, is very likely to do so. What is needed for appropriate emergent structures to arise in a mind, is mainly that the knowledge representation is sufficiently flexible to allow these structures, and the cognitive processes are sufficiently intelligent to observe these structures in the environment and then mirror them internally. Of course, it also doesn't hurt if the internal structures and processes are at least slightly *biased* toward the origination of the particular high-level emergent structures that are characteristic of the system's environment/goals; and this is indeed the case with CogPrime ... biases toward hierarchical, heterarchical, dual and mirrorhouse networks are woven throughout the system design, in a thoroughgoing though not extremely systematic way.

48.12 Ethical AGI

Creating an AGI with guaranteeably ethical behavior seems an infeasible task; but of course, no human is guaranteeably ethical either, and in fact it seems almost guaranteed that in any moderately large group of humans there are going to be some with strong propensities for extremely unethical behaviors, according to any of the standard human ethical codes. One of our motivations in developing CogPrime has been the belief that *an AGI system, if supplied with a commonsensically ethical goal system and an intentional component based on rigorous uncertain inference, should be able to reliably achieve a much higher level of commonsensically ethical behavior than any human being.*

Our explorations in the detailed design of CogPrime 's goal system have done nothing to degrade this belief. While we have not yet developed any CogPrime system to the point where experimenting with its ethics is meaningful, based on our understanding of the current design it seems to us that

- a typical CogPrime system will display a much more consistent and less conflicted and confused motivational system than any human being, due to its explicit orientation toward carrying out actions that (based on its knowledge) rationally seem most likely to lead to achievement of its goals
- if a CogPrime system is given goals that are consistent with commonsensical human ethics (say, articulated in natural language), and then educated in an ethics-friendly environment such as a virtual or physical school, then it is reasonable to expect the CogPrime system will ultimately develop an advanced (human adult level or beyond) form of commmonsensical human ethics

Human ethics is itself wracked with inconsistencies, so one cannot expect a rationality-based AGI system to precisely mirror the ethics of any particular human individual or cultural system. But given the degree to which general intelligence represents adaptation to its environment, and interpretation of natural language depends on life history and context, it seems very likely to us that a CogPrime system, if supplied with a human-commonsense-ethics based goal system and then raised by compassionate and intelligent humans in a school-type environment, would arrive at its own variant of human-commonsense-ethics. The AGI system's ethics would then interact with human ethical systems in complex ways, leading to ongoing evolution of both systems and the development of new cultural and ethical patterns. Predicting the future is difficult even in the absence of radical advanced technologies, but our intuition is that this path has the potential to lead to beneficial outcomes for both human and machine intelligence.

48.13 Toward Superhuman General Intelligence

Human-level AGI is a difficult goal, relative to the current state of scientific understanding and engineering capability, and most of this book has been focused on our ideas about how to achieve it. However, we also suspect the CogPrime architecture has the ultimate potential to push beyond the human level in many ways. As part of this suspicion we advance the claim that *once sufficiently advanced, a CogPrime system should be able to radically self-improve via a variety of methods, including supercompilation and automated theorem-proving.*

Supercompilation allows procedures to be automatically replaced with equivalent but massively more time-efficient procedures. This is particularly valuable in that it allows AI algorithms to learn new procedures without

much heed to their efficiency, since supercompilation can always improve the efficiency afterwards. So it is a real boon to automated program learning.

Theorem-proving is difficult for current narrow-AI systems, but for an AGI system with a deep understanding of the context in which each theorem exists, it should be much easier than for human mathematicians. So we envision that ultimately an AGI system will be able to design itself new algorithms and data structures via proving theorems about which ones will best help it achieve its goals in which situations, based on mathematical models of itself and its environment. Once this stage is achieved, it seems that machine intelligence may begin to vastly outdo human intelligence, leading in directions we cannot now envision.

While such projections may seem science-fictional, we note that the CogPrime architecture explicitly supports such steps. If human-level AGI is achieved within the CogPrime framework, it seems quite feasible that profoundly self-modifying behavior could be achieved fairly shortly thereafter. For instance, one could take a human-level CogPrime system and teach it computer science and mathematics, so that it fully understood the reasoning underlying its own design, and the whole mathematics curriculum leading up the the algorithms underpinning its cognitive processes.

48.13.1 Conclusion

What we have sought to do in these pages is, mainly,

- to articulate a theoretical perspective on general intelligence, according to which the creation of a human-level AGI doesn't require anything *that* extraordinary, but "merely" an appropriate combination of closely interoperating algorithms operating on an appropriate multi-type memory system, utilized to enable a system in an appropriate body and environment to figure out how to achieve its given goals
- to describe a software design (CogPrime) that, according to this somewhat mundane but theoretically quite well grounded vision of general intelligence, appears likely (according to a combination of rigorous and heuristic arguments) to be able to lead to human-level AGI using feasible computational resources
- to describe some of the preliminary lessons we've learned via implementing and experimenting with aspects of the CogPrime design, in the OpenCog system

In this concluding chapter we have focused on the "combination of rigorous and heuristic arguments" that lead us to consider it likely that CogPrime has the potential to lead to human-level AGI using feasible computational resources.

We also wish to stress that *not all of our arguments and ideas need to be 100% correct in order for the project to succeed.* The quest to create AGI is a mix of theory, engineering, and scientific and unscientific experimentation. If the current CogPrime design turns out to have significant shortcomings, yet still brings us a significant percentage of the way toward human-level AGI, the results obtained along the path will very likely give us clues about how to tweak the design to more effectively get the rest of the way there. And the OpenCog platform is extremely flexible and extensible, rather than being tied to the particular details of the CogPrime design. While we do have faith that the CogPrime design as described here has human-level AGI potential, we are also pleased to have a development strategy and implementation platform that will allow us to modify and improve the design in whatever suggestions are made by our ongoing experimentation.

Many great achievements in history have seemed more magical before their first achievement than afterwards. Powered flight and spaceflight are the most obvious examples, but there are many others such as mobile telephony, prosthetic limbs, electronically deliverable books, robotic factory workers, and so on. We now even have wireless transmission of power (one can recharge cellphones via wifi), though not yet as ambitiously as Tesla envisioned. We very strongly suspect that human-level AGI is in the same category as these various examples: an exciting and amazing achievement, which however is achievable via systematic and careful application of fairly mundane principles. We believe computationally feasible human-level intelligence is both *complicated* (involving many interoperating parts, each sophisticated in their own right) and *complex* (in the sense of involving many emergent dynamics and structures whose details are not easily predictable based on the parts of the system) ... but that neither the complication nor the complexity is an obstacle to engineering human-level AGI.

Furthermore, while ethical behavior is a complex and subtle matter for humans *or* machines, we believe that the production of human-level AGIs that are not only intelligent but also *beneficial* to humans and other biological sentiences, is something that is probably tractable to achieve based on a combination of careful AGI design and proper AGI education and "parenting." One of the motivations underlying our design has been to create an artificial mind that has broadly humanlike intelligence, yet has a more rational and self-controllable motivational system than humans, thus ultimately having the potential for a greater-than-human degree of ethical reliability alongside its greater-than-human intelligence.

In our view, what is needed to create human-level AGI is not a new scientific breakthrough, nor a miracle, but "merely" a sustained effort over a number of years by a moderate-sized team of appropriately-trained professionals, completing the implementation of the design in this book and then parenting and educating the resulting implemented system. CogPrime is by no means the only possible path to human-level AGI, but we believe it is considerably more fully thought-through and fleshed-out than any available

alternatives. Actually, we would love to see CogPrime and a dozen alternatives simultaneously pursued – this may seem ambitious, but it would cost a fraction of the money currently spent on other sorts of science or engineering, let alone the money spent on warfare or decorative luxury items. We strongly suspect that, in hindsight, our human and digital descendants will feel amazed that their predecessors allocated so few financial and attentional resources to the creation of powerful AGI, and consequently took *so long* to achieve such a fundamentally straightforward thing.

Chapter 49
Build Me Something I Haven't Seen: A CogPrime Thought Experiment

49.1 Introduction

AGI design necessarily leads one into some rather abstract spaces – but being a human-like intelligence in the everyday world is a pretty concrete thing. If the CogPrime research program is successful, it will result not just in abstract ideas and equations, but rather in real AGI robots carrying out tasks in the world, and AGI agents in virtual worlds and online digital spaces conducting important business, doing science, entertaining and being entertained by us, and so forth. With this in mind, in this final chapter we will bring the discussion closer to the concrete and everyday, and pursue a thought experiment of the form "How would a completed CogPrime system carry out this specific task?"

The task we will use for this thought-experiment is one we have used as a running example now and then in the preceding chapters. We consider the case of a robotically or virtually embodied CogPrime system, operating in a preschool type environment, interacting with a human whom it already knows and given the task of "Build me something with blocks that I haven't seen before."

This target task is fairly simple, but it is complex enough to involve essentially every one of CogPrime 's processes, interacting in a unified way.

We will consider the case of a simple interaction based on the above task where:

1. The human teacher tells the CogPrime agent "Build me something with blocks that I haven't seen before."
2. After a few false starts, the agent builds something it thinks is appropriate and says "Do you like it?"
3. The human teacher says "It's beautiful. What is it?"
4. The agent says "It's a car man" [and indeed, the construct has 4 wheels and a chassis vaguely like a car, but also a torso, arms and head vaguely like a person]

Of course, a complex system like CogPrime could carry out an interaction like this internally in many different ways, and what is roughly described here is just one among many possibilities.

First we will enumerate a number of CogPrime processes and explain some ways that each one may help CogPrime carry out the target task. Then we will give a more evocative narrative, conveying the dynamics that would occur in CogPrime while carrying out the target task, and mentioning how each of the enumerated cognitive processes as it arises in the narrative.

49.2 Roles of Selected Cognitive Processes

Now we review a number of the more interesting CogPrime cognitive processes mentioned in previous chapters of the book, for each one indicating one or more of the roles it might play in helping a CogPrime system carry out the target task. Note that this list is incomplete in many senses, e.g. it doesn't list all the cognitive processes, nor all the roles played by the ones listed. The purpose is to give an evocative sense of the roles played by the different parts of the design in carrying out the task.

- Chapter 19 (OpenCog Framework)
  - **Freezing/defrosting**.
    - When the agent builds a structure from blocks and decides it's not good enough to show off to the teacher, what does it do with the detailed ideas and thought process underlying the structure it built? If it doesn't like the structure so much, it may just leave this to the generic forgetting process. But if it likes the structure a lot, it may want to increase the VLTI (Very Long Term Importance) of the Atoms related to the structure in question, to be sure that these are stored on disk or other long-term storage, even after they're deemed sufficiently irrelevant to be pushed out of RAM by the forgetting mechanism.
    - When given the target task, the agent may decide to revive from disk the mind-states it went through when building crowd-pleasing structures from blocks before, so as to provide it with guidance.
- Chapter 22 (Emotion, Motivation, Attention and Control)
  - **Cognitive cycle**.
    - While building with blocks, the agent's cognitive cycle will be dominated by perceiving, acting on, and thinking about the blocks it is building with.

  - When interacting with the teacher, then interaction-relevant linguistic, perceptual and gestural processes will also enter into the cognitive cycle.
- **Emotion**. The agent's emotions will fluctuate naturally as it carries out the task.
  - If it has a goal of pleasing the teacher, then it will experience happiness as its expectation of pleasing the teacher increases.
  - If it has a goal of experiencing novelty, then it will experience happiness as it creates structures that are novel in its experience.
  - If it has a goal of learning, then it will experience happiness as it learns new things about blocks construction.
  - On the other hand, it will experience unhappiness as its experienced or predicted satisfaction of these goals decreases.
- **Action selection**
  - In dialoguing with the teacher, action selection will select one or more DialogueController schema to control the conversational interaction (based on which DC schema have proved most effective in prior similar situations.
    - When the agent wants to know the teacher's opinion of its construct, what is happening internally is that the "please teacher" Goal Atom gets a link of the conceptual form (Implication "find out teacher's opinion of my current construct" "please teacher"). This link may be created by PLN inference, probably largely by analogy to previously encountered similar situations. Then, GoalImportance is spread from the "please teacher" Goal Atom to the "find out teacher's opinion of my current construct" Atom (via the mechanism of sending an RFS package to the latter Atom). More inference causes a link (Implication "ask the teacher for their opinion of my current construct" "find out teacher's opinion of my current construct") to be formed, and the "ask the teacher for their opinion of my current construct" Atom to get GoalImportance also. Then PredicateSchematization causes the predicate "ask the teacher for their opinion of my current construct" to get turned into an actionable schema, which gets GoalImportance, and which gets pushed into the ActiveSchemaPool via Goal-driven action selection. Once the schema version of "ask the teacher for their opinion of my current construct" is in the ActiveSchemaPool, it then invokes natural language generation Tasks, which lead to the formulation of an English sentence such as "Do you like it?"
    - When the teacher asks "It's beautiful. What is it?", then the NL comprehension MindAgent identifies this as a question, and the "please teacher" Goal Atom gets a link of the conceptual form (Implication "answer the question the teacher just

asked" "please teacher"). This follows simply from the knowledge (Implication ("teacher has just asked a question" AND "I answer the teacher's question") ("please teacher")), or else from more complex knowledge refining this Implication. From this point, things proceed much as in the case "Do you like it?" described just above.

  - Consider a schema such as "pick up a red cube and place it on top of the long red block currently at the top of the structure" (let's call this P). Once P is placed in the ActiveSchemaPool, then it runs and generates more specific procedures, such as the ones needed to find a red cube, to move the agent's arm toward the red cube and grasp it, etc. But the execution of these specific low-level procedures is done via the ExecutionManager, analogously to the execution of the specifics of generating a natural language sentence from a collection of semantic relationships. Loosely speaking, reaching for the red cube and turning simple relationships into a simple sentences, are considered as "automated processes" not requiring holistic engagement of the agent's mind. What the generic, more holistic Action Selection mechanism does in the present context is to figure out to put P in the ActiveSchemaPool in the first place. This occurs because of a chain such as: P predictively implies (with a certain probabilistic weight) "completion of the car-man structure", which in turn predictively implies "completion of a structure that is novel to the teacher," which in turn predictively implies "please the teacher," which in turn implies "please others," which is assumed an Ubergoal (a top-level system goal).

- **Goal Atoms**. As the above items make clear, the scenario in question requires the initial Goal Atoms to be specialized, via the creation of more and more particular subgoals suiting the situation at hand.
- **Context Atoms**.
  - Knowledge of the context the agent is in can help it disambiguate language it hears, e.g. knowing the context is blocks-building helps it understand which sense of the word "blocks" is meant.
  - On the other hand, if the context is that the teacher is in a bad mood, then the agent might know via experience that in this context, the strength of (Implication "ask the teacher for their opinion of my current construct" "find out teacher's opinion of my current construct") is lower than in other contexts.
- **Context formation**.
  - A context like blocks-building or "teacher in a bad mood" may be formed by clustering over multiple experience-sets, i.e. forming Atoms that refer to spatiotemporally grouped sets of percepts/concepts/actions, and grouping together similar Atoms of this nature into clusters.

  - The Atom referring to the cluster of experience-sets involving blocks-building will then survive as an Atom if it gets involved in relationships that are important or have surprising truth values. If many relationships have significantly different truth-value inside the blocks-building context than outside it, this means it's likely that the blocks-building ConceptNode will remain as an Atom with reasonably high LTI, so it can be used as a context in future.
- **Time-dependence of goals**. Many of the agent's goals in this scenario have different importances over different time scales. For instance "please the teacher" is important on multiple time-scales: the agent wants to please the teacher in the near term but also in the longer term. But a goal like "answer the question the teacher just asked" has an intrinsic time-scale to it; if it's not fulfilled fairly rapidly then its importance goes away.

- Chapter 23 (Attention allocation)
  - **ShortTermImportance versus LongTermImportance**. While conversing, the concepts and immediately involved in the conversation (including the Atoms describing the agents in the conversation) have very high STI. While building, Atoms representing to the blocks and related ideas about the structures being built (e.g. images of cars and people perceived or imagined in the past) have very high STI. But the reason these Atoms are in RAM prior to having their STI boosted due to their involvement in the agent's activities, is because they had their LTI boosted at some point in the past. And after these Atoms leave the AttentionalFocus and their STI reduces, they will have boosted LTI and hence likely remain in RAM for a long while, to be involved in "background thought", and in case they're useful in the AttentionalFocus again.
  - **HebbianLink formation** As a single example, the car-man has both wheels and arms, so now a Hebbian association between wheels and arms will exist in the agent's memory, to potentially pop up again and guide future thinking. The very idea of a car-man likely emerged partly due to previously formed HebbianLinks – because people were often seen sitting in cars, the association between person and car existed, which made the car concept and the human concept natural candidates for blending.
  - **Data mining the System Activity Table**. The HebbianLinks mentioned above may have been formed via mining the SystemActivityTable
  - **ECAN based associative memory**. When the agent thinks about making a car, this spreads importance to various Atoms related to the car concept, and one thing this does is lead to the emergence of the car attractor into the AttentionalFocus. The different aspects of a car are represented by heavily interlinked Atoms, so that when

some of them become important, there's a strong tendency for the others to also become important – and for "car" to then emerge as an attractor of importance dynamics.

- **Schema credit assignment**.
  - Suppose the agent has a subgoal of placing a certain blue block on top of a certain red block. It may use a particular motor schema for carrying out this action – involving, for instance, holding the blue block above the red block and then gradually lowering it. If this schema results in success (rather than in, say, knocking down the red block), then it should get rewarded via having its STI and LTI boosted and also having the strength of the link between it and the subgoal increased.
  - Next, suppose that a certain cognitive schema (say, the schema of running multiple related simulations and averaging the results, to estimate the success probability of a motor procedure) was used to arrive at the motor schema in question. Then this cognitive schema may get passed some importance from the motor schema, and it will get the strength of its link to the goal increased. In this way credit passes backwards from the goal to the various schema directly or indirectly involved in fulfilling it.
- **Forgetting**. If the agent builds many structures from blocks during its lifespan, it will accumulate a large amount of perceptual memory.

- Chapter 24 (Goal and Action Selection). Much of the use of the material in this chapter was covered above in the bullet point for Chapter 22, but a few more notes are:
  - **Transfer of RFS between goals**. Above it was noted that the link (Implication "ask the teacher for their opinion of my current construct" "find out teacher's opinion of my current construct") might be formed and used as a channel for GoalImportance spreading.
  - **Schema Activation**. Supposing the agent is building a man-car, it may have car-building schema and man-building schema in its ActiveSchemaPool at the same time, and it may enact both of them in an interleaved manner. But if each tend to require two hands for their real-time enaction, then schema activation will have to pass back and forth between the two of them, so that at any one time, one is active whereas the other one is sitting in the ActiveSchemaPool waiting to get activated.
  - **Goal Based Schema Learning**. To take a fairly low-level example, suppose the agent has the (sub)goal of making an arm for a blocks-based person (or man-car), given the presence of a blocks-based torso. Suppose it finds a long block that seems suitable to be an arm. It then has the problem of figuring out how to attach the arm to the body. It may try out several procedures in its internal simulation world, until it finds one that works: *hold the arm in the right position while one*

end of it rests on top of some block that is part of the torso, then place some other block on top of that end, then slightly release the arm and see if it falls. If it doesn't fall, leave it. If it seems about to fall, then place something heavier atop it, or shove it further in toward the center of the torso. The procedure learning process could be MOSES here, or it could be PLN.

- Chapter 25 (Procedure Evaluation)
  - **Inference Based Procedure Evaluation**. A procedure for man-building such as "first put up feet, then put up legs, then put up torso, then put up arms and head" may be synthesized from logical knowledge (via predicate schematization) but without filling in the details of how to carry out the individual steps, such as "put up legs." If a procedure with abstract (ungrounded) schema like PutUpTorso is chosen for execution and placed into the ActiveSchemaPool, then in the course of execution, inferential procedure evaluation must be used to figure out how to make the abstract schema actionable. The GoalDrivenActionSelection MindAgent must make the choice whether to put a not-fully-grounded schema into the ActiveSchemaPool, rather than grounding it first and then making it active; this is the sort of choice that may be made effectively via learned cognitive schema.
- Chapter 26 (Perception and Action)
  - **ExperienceDB**. No person remembers every blocks structure they ever saw or built, except maybe some autists. But a CogPrime can store all this information fairly easily, in its ExperienceDB, even if it doesn't keep it all in RAM in its AtomSpace. It can also store everything anyone ever said about blocks structures in its vicinity.
  - **Perceptual Pattern Mining**
  - **Object Recognition**. Recognizing structures made of blocks as cars, people, houses, etc. requires fairly abstract object recognition, involving identifying the key shapes and features involved in an object-type, rather than just going by simple visual similarity.
  - **Hierarchical Perception Networks**. If the room is well-lit, it's easy to visually identify individual blocks within a blocks structure. If the room is darker, then more top-down processing may be needed – identifying the overall shape of the blocks structure may guide one in making out the individual blocks.
  - **Hierarchical Action Networks**. Top-down action processing tells the agent that, if it wants to pick up a block, it should move its arm in such a way as to get its hand near the block, and then move its hand. But if it's still learning how to do that sort of motion, more likely it will do this, but then start moving its its hand and find that it's hard to get a grip on the block – and then have to go back and

move its arm a little differently. Iterating between broader arm/hand movements and more fine-grained hand/finger movements is an instance of information iteratively passing up and down an hierarchical action network.

- **Coupling of Perception and Action Networks**. Picking up a block in the dark is a perfect example of rich coupling of perception and action networks. Feeling the block with the fingers helps with identifying blocks that can't be clearly seen.

- Chapter 30 (Procedure Learning)
  - **Specification Based Procedure Learning**.
    - Suppose the agent has never seen a horse, but the teacher builds a number of blocks structures and calls them horses, and draws a number of pictures and calls them horses. This may cause a procedure learning problem to be spawned, where the fitness function is accuracy at distinguishing horses from non-horses.
    - Learning to pick up a block is specification-based procedure learning, where the specification is to pick up the block and grip it and move it without knocking down the other stuff near the block.
  - **Representation Building**.
    - In the midst of building a procedure to recognize horses, MOSES would experimentally vary program nodes recognizing visual features into other program nodes recognizing other visual features
    - In the midst of building a procedure to pick up blocks, MOSES would experimentally vary program nodes representing physical movements into other nodes representing physical movements
    - In both of these cases, MOSES would also carry out the standard experimental variations of mathematical and control operators according to its standard representation-building framework

- Chapter 31 (Imitative, Reinforcement and Corrective Learning)
  - **Reinforcement Learning**.
    - Motor procedures for placing blocks (in simulations or reality) will get rewarded if they don't result in the blocks structure falling down, punished otherwise.
    - Procedures leading to the teacher being pleased, in internal simulations (or in repeated trials of scenarios like the one under consideration), will get rewarded; procedures leading to the teacher being displeased will get punished.
  - **Imitation Learning**. If the agent has seen others build with blocks before, it may summon these memories and then imitate the actions it has seen others take.
  - **Corrective Learning**. This would occur if the teacher intervened in the agent's block-building and guided him physically – e.g. steadying

his shaky arm to prevent him from knocking the blocks structure over.

- Chapter 32 (Hillclimbing)
  - **Complexity Penalty**. In learning procedures for manipulating blocks, the complexity penalty will militate against procedures that contain extraneous steps.
- Chapter 33 (Probabilistic Evolutionary Procedure Learning)
  - **Supplying Evolutionary Learning with Long-Term Memory**. Suppose the agent has previously built people from clay, but never from blocks. It may then have learned a "classification model" predicting which clay people will look appealing to humans, and which won't. It may then transfer this knowledge, using PLN, to form a classification model predicting which blocks-people will look appealing to humans, and which won't.
  - **Fitness Function Estimation via Integrative Intelligence**. To estimate the fitness of a procedure for, say, putting an arm on a blocks-built human, the agent may try out the procedure in the internal simulation world; or it may use PLN inference to reason by analogy to prior physical situations it's observed. These allow fitness to be estimated without actually trying out the procedure in the environment.
- Chapter 34 (Probabilistic Logic Networks)
  - **Deduction**. This is a tall skinny structure; tall skinny structures fall down easily; thus this structure may fall down easily.
  - **Induction**. This teacher is talkative; this teacher is friendly; therefore the talkative are generally friendly.
  - **Abduction**. This structure has a head and arms and torso; a person has a head and arms and torso; therefore this structure is a person.
  - **PLN forward chaining**. What properties might a car-man have, based on inference from the properties of cars and the properties of men?
  - **PLN backward chaining**.
    - An inference target might be: *Find X so that X looks something like a wheel and can be attached to this blocks-chassis, and I can find four fairly similar copies.*
    - Or: *Find the truth value of the proposition that this structure looks like a car.*
  - **Indefinite truth values**. Consider the deductive inference "This is a tall skinny structure; tall skinny structures fall down easily; thus this structure may fall down easily." In this case, the confidence of the second premise may be greater than the confidence of the first premise, which may result in an intermediate confidence for the conclusion,

according to the propagation of indefinite probabilities through the PLN deduction rule.
  - **Intensional inference**. Is the blocks-structure a person? According to the definition of intensional inheritance, it shares many informative properties with people (e.g. having arms, torso and head), so to a significant extent, it is a person.
  - **Confidence decay**. The agent's confidence in propositions regarding building things with blocks should remain nearly constant. The agent's confidence in propositions regarding the teacher's taste should decay more rapidly. This should occur because the agent should observe that, in general, propositions regarding physical object manipulation tend to retain fairly constant truth value, whereas propositions regarding human tastes tend to have more rapidly decaying truth value.

- Chapter 35 (Spatiotemporal Inference)
  - **Temporal reasoning**. Suppose, after the teacher asks "What is it?", the agent needs to think a while to figure out a good answer. But maybe the agent knows that it's rude to pause too long before answering something to a direct question. Temporal reasoning helps figure out "how long is too long" to wait before answering.
  - **Spatial reasoning**. Suppose the agent puts shoes on the wheels of the car. This is a joke relying on the understanding that wheels hold a car up, whereas feet hold a person up, and the structure is a car-man. But it also relies on the spatial inferences that: the car's wheels are in the right position for the man's feet (below the torso); and, the wheels are below the car's chassis just like a person's feet are below its torso.

- Chapter 36 (Inference Control)
  - **Evaluator Choice as a Bandit Problem**. In doing inference regarding how to make a suitably humanlike arm for the blocks-man, there may be a choice between multiple inference pathways, perhaps one that relies on analogy to other situations building arms, versus one that relies on more general reasoning about lengths and weights of blocks. The choice between these two pathways will be made randomly with a certain probabilistic bias assigned to each one, via prior experience.
  - **Inference Pattern Mining**. The probabilities used in choosing which inference path to take, are determined in part by prior experience – e.g. maybe it's the case that in prior situations of building complex blocks structures, analogy has proved a better guide than naive physics, thus the prior probability of the analogy inference pathway will be nudged up.

  - **PLN and Bayes Nets**. What's the probability that the blocksman's hat will fall off if the man-car is pushed a little bit to simulate driving? This question could be resolved in many ways (e.g. by internal simulation), but one possibility is inference. If this is resolved by inference, it's the sort of conditional probability calculation that could potentially be done faster if a lot of the probabilistic knowledge from the AtomSpace were summarized in a Bayes Net. Updating the Bayes net structure can be slow, so this is probably not appropriate for knowledge that is rapidly shifting; but knowledge about properties of blocks structures may be fairly persistent after the agent has gained a fair bit of knowledge by playing with blocks a lot.

- Chapter 37 (Pattern Mining)
  - **Greedy Pattern Mining**.
    - "Push a tall structure of blocks and it tends to fall down" is the sort of repetitive pattern that could easily be extracted from a historical record of perceptions and (the agent's and others') actions via simple greedy pattern mining algorithm.
    - If there is a block that is shaped like a baby's rattle, with a long slender handle and then a circular shape at the end, then greedy pattern mining may be helpful due to having recognized the pattern that structures like this are sometimes rattles – and also that structures like this are often stuck together, with the handle part connected sturdily to the circular part.
  - **Evolutionary Pattern Mining** "Push a tall structure of blocks with a wide base and a gradual narrowing toward the top and it may not fall too badly" is a more complex pattern that may not be found via greedy mining, unless the agent has dealt with a lot of pyramids. But

- Chapter 38 (Concept Formation)
  - **Formal Concept Analysis**. Suppose there are many long, slender blocks of different colors and different shapes (some cylindrical, some purely rectangular for example). Learning this sort of concept based on common features is exactly what FCA is good at (and when the features are defined fuzzily or probabilistically, it's exactly what uncertain FCA is good at). Learning the property of "slender" itself is another example of something uncertain FCA is good at – it would learn this if there were many concepts that preferentially involved slender things (even though formed on the basis of concepts other than slenderness)
  - **Conceptual Blending** . The concept of a "car-man" or "man-car" is an obvious instance of conceptual blending. The agents know that building a man won't surprise the teacher, and nor will building a car ... but both "man" and "car" may pop to the forefront of its mind

(i.e. get a briefly high STI) when it thinks about what to build. But since it knows it has to do something new or surprising, there may be a cognitive schema that boosts the amount of funds to the ConceptBlending MindAgent, causing it to be extra-active. In any event, the ConceptBlending agent seeks to find ways to combine important concepts; and then PLN explores these to see which ones may be able to achieve the given goal of surprising the teacher (which includes subgoals such as actually being buildable).

- Chapter 39 (Dimensional Embedding)
  - **Dimensional Embedding** . When the agent needs to search its memory for a previously seen blocks structure similar to the currently observed one – or for a previously articulated thought similar to the one it's currently trying to articulate – then it needs to to a search through its large memory for "an entity similar to X" (where X is a structure or a thought). This kind of search can be quite computationally difficult – but if the entities in question have been projected into an embedding space, then it's quite rapid. (The cost is shifted to the continual maintenance of the embedding space, and its periodic updating; and there is some error incurred in the projection, but in many cases this error is not a show-stopper.)
  - **Embedding Based Inference Control** . Rapid search for answers to similarity or inheritance queries can be key for guiding inference in appropriate directions; for instance reasoning about how to build a structure with certain properties, can benefit greatly from rapid search for previously-encountered substructures currently structurally or functionally similar to the substructures one desires to build.
- Chapter 40 (Simulation and Episodic Memory)
  - **Fitness Estimation via Simulation** . One way to estimate whether a certain blocks structure is likely to fall down or not, is to build it in one's "mind's eye" and see if the physics engine in one's mind's-eye causes it to fall down. This is something that in many cases will work better for CogPrime than for humans, because CogPrime has a more mathematically accurate physics engine than the human mind does; however, in cases that rely heavily on naive physics rather than, say, direct applications of Newton's Laws, then CogPrime 's simulation engine may underperform the typical human mind.
  - **Concept Formation via Simulation** . Objects may be joined into categories using uncertain FCA, based on features that they are identified to have via "simulation experiments" rather than physical world observations. For instance, it may be observed that pyramid-shaped structures fall less easily than pencil-shaped tower structures – and the concepts corresponding to these two categories may be formed

  - from experiments run in the internal simulation world, perhaps inspired by isolated observations in the physical world.
  - **Episodic Memory** . Previous situations in which the agent has seen similar structures built, or been given similar problems to solve, may be brought to mind as "episodic movies" playing in the agent's memory. By watching what happens in these replayed episodic movies, the agent may learn new declarative or procedural knowledge about what to do. For example, maybe there was some situation in the agent's past where it saw someone asked to do something surprising, and that someone created something funny. This might (via a simple PLN step) bias the agent to create something now, which it has reason to suspect will cause others to laugh.

- Chapter 41 (Integrative Procedure Learning)
  - **Concept-Driven Procedure Learning** . Learning the concept of "horse", as discussed above in the context of Chapter 30, is an example of this
  - **Predicate Schematization** . The synthesis of a schema for manbuilding, as discussed above in the context of Chapter 25 , is an example of this

- Chapter 42 (Map Formation)
  - **Map Formation** . The notion of a car involves many aspects: the physical appearance of cars, the way people get in and out of cars, the ways cars drive, the noises they make, etc. All these aspects are represented by Atoms that are part of the car map, and are richly interconnected via HebbianLinks as well as other links.
  - **Map Encapsulation** . The car map forms implicitly via the interaction of multiple cognitive dynamics, especially ECAN. But then the MapEncapsulation MindAgent may do its pattern mining and recognize this map explicitly, and form a PredicateNode encapsulating it. This PredicateNode may then be used in PLN inference, conceptual blending, and so forth (e.g. helping with the formation of a concept like car-man via blending).

- Chapter 44 (Natural Language Comprehension)
  - **Experience Based Disambiguation** . The particular dialogue involved in the present example doesn't require any nontrivial word sense disambiguation. But it does require parse selection, and semantic interpretation selection:
    - In "Build me something with blocks," the agent has no trouble understanding that "blocks" means "toy building blocks" rather than, say, "city blocks", based on many possible mechanisms, but most simply importance spreading

  - "Build me something with blocks" has at least three interpretations: the building could be carried out using blocks with a tool; or the thing built could be presented alongside blocks; or the thing built could be composed of blocks. The latter is the most commonsensical interpretation for most humans, but that is because we have heard the phrase "building with blocks" used in a similarly grounded way before (as well as other similar phrases such as "playing with Legos", etc., whose meaning helps militate toward the right interpretation via PLN inference and importance spreading). So here we have a simple example of experience-based disambiguation, where experiences at various distances of association from the current one are used to help select the correct parse.
  - A subtler form of semantic disambiguation is involved in interpreting the clause "that I haven't seen before." A literal-minded interpretation would say that this requirement is fulfilled by any blocks construction that's not precisely identical to one the teacher has seen before. But of course, any sensible human knows this is an idiomatic clause that means "significantly different from anything I've seen before." This could be determined by the CogPrime agent if it has heard the idiomatic clause before, or if it's heard a similar idiomatic phrase such as "something I've never done before." Or, even if the agent has never heard such an idiom before, it could potentially figure out the intended meaning simply because the literal-minded interpretation would be a pointless thing for the teacher to say. So if it knows the teacher usually doesn't add useless modificatory clauses onto their statements, then potentially the agent could guess the correct meaning of the phrase.

- Chapter 45 (Language Generation)
  - **Experience-Based Knowledge Selection for Language Generation** . When the teacher asks "What is it?", the agent must decide what sort of answer to give. Within the confines of the QuestionAnswering DialogueController, the agent could answer "A structure of blocks", or "A part of the physical world", or "A thing", or "Mine." (Or, if it were running another DC, it could answer more broadly, e.g. "None of your business," etc.). However, the QA DC tells it that, in the present context, the most likely desired answer is one that the teacher doesn't already know; and the most important property of the structure that the teacher doesn't obviously already know is the fact that it depicts a "car man." Also, memory of prior conversations may bring up statements like "It's a horse" in reference to a horse built of blocks, or a drawing of a horse, etc.

  - **Experience-Based Guidance of Word and Syntax Choice** . The choice of phrase "car man" requires some choices to be made. The agent could just as well say "It's a man with a car for feet" or "It's a car with a human upper body and head" or "It's a car centaur," etc. A bias toward simple expressions would lead to "car man." If the teacher were known to prefer complex expressions, then the agent might be biased toward expressing the idea in a different way.

- Chapter 47 (Natural Language Dialogue)
  - **Adaptation of Dialogue Controllers** . The QuestionAsking and QuestionAnswering DialogueControllers both get reinforcement from this interaction, for the specific internal rules that led to the given statements being made.

49.3 A Semi-Narrative Treatment

Now we describe how a CogPrime system might carry out the specified task in a semi-narrative form, weaving in the material from the previous section as we go along, and making some more basic points as well. The semi-narrative covers most but not all of the bullet points from the previous section, but with some of the technical details removed; and it introduces a handful of new examples not given in the bullet points.

The reason this is called a semi-narrative rather than a narrative is that there is no particular linear order to the processes occurring in each phase of the situation described here. CogPrime 's internal cognitive processes do not occur in a linear narrative; rather, what we have is a complex network of interlocking events. But still, describing some of these events concretely in a manner correlated with the different stages of a simple interaction, may have some expository value.

The human teacher tells the CogPrime agent "Build me something with blocks that I haven't seen before."

Upon hearing this, the agent's cognitive cycles are dominated by language processing and retrieval from episodic and sensory memory.

The agent may decide to revive from disk the mind-states it went through when building human-pleasing structures from blocks before, so as to provide it with guidance

It will likely experience the emotion of happiness, because it anticipates the pleasure of getting rewarded for the task in future.

The ubergoal of pleasing the teacher gets active (gets funded significantly with STI currency), as there it becomes apparent there are fairly clear ways of fulfilling that goal (via the subgoal S of building blocks structures that will get positive response from the teacher). Other ubergoals like gaining

knowledge are not funded as much with STI currency just now, as they are not immediately relevant.

Action selection, based on ImplicationLinks derived via PLN (between various possible activities and the subgoal S) causes it to start experimentally building some blocks structures. Past experience with building (turned into ImplicationLinks via mining the SystemActivityTable) tells it that it may want to build a little bit in its internal simulation world before building in the external world, causing STI currently to flow to the simulation MindAgent.

The Atom corresponding to the context blocks-building gets high STI and is pushed into the AttentionalFocus, making it likely that many future inferences will occur in this context. Other Atoms related to this one also get high STI (the ones in the blocks-building map, and others that are especially related to blocks-building in this particular context).

After a few false starts, the agent builds something it thinks is appropriate and says "Do you like it?"

Now that the agent has decided what to do to fulfill its well-funded goal, its cognitive cycles are dominated by action, perception and related memory access and concept creation.

An obvious subgoal is spawned: build a new structure now, and make this particular structure under construction appealing and novel to the teacher. This subgoal has a shorter time scale than the high level goal. The subgoal gets some currency from its supergoal using the mechanism of RFS spreading.

Action selection must tell it when to continue building the same structure and when to try a new one, as well as more micro level choices

Atoms related to the currently pursued blocks structure get high STI.

After a failed structure (a ÒfalseÓ) is disassembled, the corresponding Atoms lose STI dramatically (leaving AF) but may still have significant LTI, so they can be recalled later as appropriate. They may also have VLTI so they will be saved to disk later on if other things push them out of RAM due to getting higher LTI.

Meanwhile everything that's experienced from the external world goes into the ExperienceDB.

Atoms representing different parts of aspects of the same blocks structure will get Hebbian links between them, which will guide future reasoning and importance spreading.

Importance spreading helps the system go from an idea for something to build (say, a rock or a car) to the specific plans and ideas about how to build it, via increasing the STI of the Atoms that will be involved in these plans and ideas.

If something apparently good is done in building a blocks structure, then other processes and actions that helped lead to or support that good thing, get passed some STI from the Atoms representing the good thing, and also may get linked to the Goal Atom representing ÒgoodÓ in this context. This leads to reinforcement learning.

The agent may play with building structures and then seeing what they most look like, thus exercising abstract object recognition (that uses procedures learned by MOSES or hillclimbing, or uncertain relations learned by inference, to guess what object category a given observed collection of percepts most likely falls into).

Since the agent has been asked to come up with something surprising, it knows it should probably try to formulate some new concepts. Because it has learned in the past, via SystemActivityTable mining, that often newly formed concepts are surprising to others. So, more STI currency is given to concept formation MindAgents, such as the ConceptualBlending Mind Agent (which, along with a lot of stuff that gets thrown out or stored for later use, comes up with ÒcarManÓ).

When the notion of ÒcarÓ is brought to mind, the distributed map of nodes corresponding to ÒcarÓ get high STI. When car-man is formed, it is reasoned about (producing new Atoms), but it also serves as a nexus of importance-spreading, causing the creation of a distributed car-man map.

If the goal of making an arm for a man-car occurs, then goal-driven schema learning may be done to learn a procedure for arm-making (where the actual learning is done by MOSES or hill-climbing).

If the agent is building a man-car, it may have man-building and car-building schema in its ActiveSchemaPool at the same time, and SchemaActivation may spread back and forth between the different modules of these two schema.

If the agent wants to build a horse, but has never seen a horse made of blocks (only various pictures and movies of horses), it may uses MOSES or hillclimbing internally to solve the problem of creating a horse-recognizer or a horse-generator which embodies appropriate abstract properties of horses. Here as in all cases of procedure learning, a complexity penalty rewards simpler programs, from among all programs that approximately fulfill the goals of the learning process.

If a procedure being executed has some abstract parts, then these may be executed by inferential procedure evaluation (which makes the abstract parts concrete on the fly in the course of execution).

To guess the fitness of a procedure for doing something (say, building an arm or recognizing a horse), inference or simulation may be used, as well as direct evaluation in the world.

Deductive, inductive and abductive PLN inference may be used in figuring out what a blocks structure will look or act like like before building it (it's tall and thin so it may fall down; it won't be bilaterally symmetric so it won't look much like a person; etc.)

Backward-chaining inference control will help figure out how to assemble something matching a certain specification Ð e.g. how to build a chassis based on knowledge of what a chassis looks like. Forward chaining inference (critically including intensional relationships) will be used to estimate the properties that the teacher will perceive a given specific structure to have.

Spatial and temporal algebra will be used extensively in this reasoning, within the PLN framework.

Coordinating different parts of the body Ð say an arm and a hand Ð will involve importance spreading (both up and down) within the hierarchical action network, and from this network to the hierarchical perception network and the heterarchical cognitive network.

In looking up Atoms in the AtomSpace, some have truth values whose confidences have decayed significantly (e.g. those regarding the teacher's tastes), whereas others have confidences that have hardly decayed at all (e.g. those regarding general physical properties of blocks).

Finding previous blocks structures similar to the current one (useful for guiding building by analogy to past experience) may be done rapidly by searching the system's internal dimensional-embedding space.

As the building process occurs, patterns mined via past experience (tall things often fall down) are used within various cognitive processes (reasoning, procedure learning, concept creation, etc.); and new pattern mining also occurs based on the new observations made as different structures are build and experimented with and destroyed.

Simulation of teacher reactions, based on inference from prior examples, helps with the evaluation of possible structures, and also of procedures for creating structures.

As the agent does all this, it experiences the emotion of curiosity (likely among other emotions), because as it builds each new structure it has questions about what it will look like and how the teacher would react to it.

The human teacher says "It's beautiful. What is it?" The agent says "It's a car man"

Now that the building is done and the teacher says something, the agent's cognitive cycles are dominated by language understanding and generation. The Atom representing the context of talking to the teacher gets high STI, and is used as the context for many ensuing inferences.

Comprehension of "it" uses anaphor resolution based on a combination of ECAN and PLN inference based on a combination of previously interpreted language and observation of the external world situation.

The agent experiences the emotion of happiness because the teacher has called its creation beautiful, which is recognizes as a positive evaluation Ð so the agent knows one of its ubergoals ("please the teacher") has been significantly fulfilled.

The goal of pleasing the teacher causes the system to want to answer the question. So the QuestionAnswering DialogueController schema gets paid a lot and gets put into the ActiveSchemaPool. In reaction to the question asked, this DC chooses a semantic graph to speak, then invokes NL generation to say it.

NL generation chooses the most compact expression that seems to adequately convey the intended meaning, so it decides on "car man" as the best

simple verbalization to match the newly created conceptual blend that it thinks effectively describes the newly created blocks structure.

The positive feedback from the user leads to reinforcement of the Atoms and processes that led to the construction of the blocks structure that has been judged beautiful (via importance spreading and SystemActivityTable mining).

49.4 Conclusion

The simple situation considered in this chapter is complex enough to involve nearly all the different cognitive processes in the CogPrime system – and many interactions between these processes. This fact illustrates one of the main difficulties of designing, building and testing an artificial mind like CogPrime – until nearly all of the system is build and made to operate in an integrated way, it's hard to do any meaningful test of the system. Testing PLN or MOSES or conceptual blending in isolation may be interesting computer science, but it doesn't tell you much about CogPrime as a design for a thinking machine.

According to the CogPrime approach, getting a simple child-like interaction like "build me something with blocks that I haven't seen before" to work properly requires a holistic, integrated cognitive system. Once one has built a system capable of this sort of simple interaction then, according to the theory underlying CogPrime , one is not that far from a system with adult human-level intelligence. Of course there will be a lot of work to do to get from a child-level system to an adult-level system – it won't necessarily unfold as "automatically" as seems to happen with a human child, because CogPrime lacks the suite of developmental processes and mechanisms that the young human brain has. But still, a child CogPrime mind capable of doing the things outlined in this chapter will have all the basic components and interactions in place, all the ones that are needed for a much more advanced artificial mind.

Of course, one could concoct a narrow-AI system carrying out the specific activities described in this chapter, much more simply than one could build a CogPrime system capable of doing these activities. But that's not the point – the point of this chapter is not to explain how to achieve some particular narrow set of activities "by any means necessary", but rather to explain how these activities might be achieved within the CogPrime framework, which has been designed with much more generality in mind.

It would be worthwhile to elaborate a number of other situations similar to the one described in this chapter, and to work through the various cognitive processes and structures in CogPrime carefully in the context of each of these situations. In fact this sort of exercise has frequently been carried out informally in the context of developing CogPrime . But this book is al-

ready long enough, so we will end here, and leave the rest for future works – emphasizing that it is via intimate interplay between concrete considerations like the ones presented in this chapter, and general algorithmic and conceptual considerations as presented in most of the chapters of this book, that we have the greatest hope of creating advanced AGI. The value of this sort of interplay actually follows from the theory of real-world general intelligence presented in Part 1 of the book. Thoroughly general intelligence is only possible given unrealistic computational resources, so real-world general intelligence is about achieving high generality given limited resources relative to the specific classes of environments relevant to a given agent. Specific situations like building surprising things with blocks are particularly important insofar as they embody broader information about the classes of environments relevant to broadly *human-like* general intelligence.

No doubt, once a CogPrime system is completed, the specifics of its handling of the situation described here will differ somewhat from the treatment presented in this chapter. Furthermore, the final CogPrime system may differ algorithmically and structurally in some respects from the specifics given in this book – it would be surprising if the process of building, testing and interacting with CogPrime didn't teach us some new things about various of the topics covered. But our conjecture is that, if sufficient effort is deployed appropriately, then a system much like the CogPrime system described in this book will be able to handle the situation described in this chapter in a roughly similar manner to the one described in this chapter – and that this will serve as a natural precursor to much more dramatic AGI achievements.

Appendix A
Glossary

: :

This glossary clarifies the meaning of various terms as they are commonly used in the context of OpenCog & CogPrime

. It doesn't attempt to provide general definitions of the terms involved; it is intentionally highly OpenCog/CogPrime

-specific. Where terms are used in multiple ways within the OpenCog/CogPrime

sphere, more than one meaning is given.

- **Abduction**: A general form of inference that goes from data describing something to a hypothesis that accounts for the data. Often in an OpenCog context, this refers to the PLN abduction rule, a specific First-Order PLN rule (If A implies C, and B implies C, then maybe A is B), which embodies a simple form of abductive inference. But OpenCog may also carry out abduction, as a general process, in other ways.
- **Action Selection**: The process via which the OpenCog system chooses which Schema to enact, based on its current goals and context.
- **Active Schema Pool**: The set of Schema currently in the midst of Schema Execution.
- **Adaptive Inference Control**: Algorithms or heuristics for guiding PLN inference, that cause inference to be guided differently based on the context in which the inference is taking place, or based on aspects of the inference that are noted as it proceeds.
- **AGI Preschool**: A virtual world or robotic scenario roughly similar to the environment within a typical human preschool, intended for AGIs to learn in via interacting with the environment and with other intelligent agents
- **Atom**: The basic entity used in OpenCog as an element for building representations. Some Atoms directly represent patterns in the world or mind, others are components of representations. There are two kinds of Atoms: Nodes and Links.

- **Atom, Frozen**: See Atom, Saved
- **Atom, Realized**: An Atom that exists in RAM at a certain point in time
- **Atom, Saved**: An Atom that has been saved to disk or other similar media, and is not actively being processed
- **Atom, Serialized**: An Atom that is serialized for transmission from one software process to another, or for saving to disk, etc.
- **Atom2Link**: A part of OpenCogPrime
  s language generation system, that transforms appropriate Atoms into words connected via link parser link types.
- **Atomspace**: A collection of Atoms, comprising the central part of the memory of an OpenCog instance.
- **Attention**: The aspect of an intelligent system's dynamics focused on guiding which aspects of an OpenCog system's memory & functionality gets more computational resources at a certain point in time
- **Attention Allocation**: The cognitive process concerned with managing the parameters and relationships guiding what the system pays attention to, at what points in time. This is a term inclusive of Importance Updating and Hebbian Learning.
- **Attentional Currency**: Short Term Importance and Long Term Importance values are implemented in terms of two different types of artificial money, STICurrency and LTICurrency. Theoretically these may be converted to one another.
- **Attentional Focus**: The Atoms in an OpenCog Atomspace whose ShortTermImportance values lie above a critical threshold (the AttentionalFocus Boundary). The Attention Allocation subsystem treats these Atoms differently. Qualitatively, these Atoms constitute the system's main focus of attention during a certain interval of time, i.e. it's moving bubble of attention.
- **Attentional Memory**: A system's memory of what it's useful to pay attention to, in what contexts. In CogPrime this is managed by the attention allocation subsystem.
- **Backward Chainer**: A piece of software, wrapped in a MindAgent, that carries out backward chaining inference using PLN
- **CIM-Dynamic**: Concretely-Implemented Mind Dynamic, a term for a cognitive process that is implemented explicitly in OpenCog (as opposed to allowed to emerge implicitly from other dynamics). Sometimes a CIM-Dynamic will be implemented via a single MindAgent, sometimes via a set of multiple interrelated MindAgents, occasionally by other means.
- **Cognition**: In an OpenCog context, this is an imprecise term. Sometimes this term means any process closely related to intelligence; but more often it's used specifically to refer to more abstract reasoning/learning/etc, as distinct from lower-level perception and action.
- **Cognitive Architecture**: This refers to the logical division of an AI system like OpenCog into interacting parts and processes representing

different conceptual aspects of intelligence. It's different from the software architecture, though of course certain cognitive architectures and certain software architectures fit more naturally together.

- **Cognitive Cycle**: The basic "loop" of operations that an OpenCog system, used to control an agent interacting with a world, goes through rapidly each "subjective moment." Typically a cognitive cycle should be completed in a second or less. It minimally involves perceiving data from the world, storing data in memory, and deciding what if any new actions need to be taken based on the data perceived. It may also involve other processes like deliberative thinking or metacognition. Not all OpenCog processing needs to take place within a cognitive cycle.
- **Cognitive Schematic**: An implication of the form "Context AND Procedure IMPLIES goal". Learning and utilization of these is key to CogPrime 's cognitive process.
- **Cognitive Synergy**: The phenomenon by which different cognitive processes, controlling a single agent, work together in such a way as to help each other be more intelligent. Typically, if one has cognitive processes that are individually susceptible to combinatorial explosions, cognitive synergy involves coupling them together in such a way that they can help one another over come each other's internal combinatorial explosions. The CogPrime design is reliant on the hypothesis that its key learning algorithms will display dramatic cognitive synergy when utilized for agent control in appropriate environments.
- **CogPrime** : The name for the AGI design presented in this book, which is designed specifically for implementation within the OpenCog software framework (and this implementation is OpenCogPrime)
- **CogServer**: A piece of software, within OpenCog, that wraps up an Atomspace and a number of MindAgents, along with other mechanisms like a Scheduler for controlling the activity of the MindAgents, and code for important and exporting data from the Atomspace.
- **Cognitive Equation**: The principle, identified in Ben Goertzel's 1994 book "Chaotic Logic, "that minds are collections of pattern-recognition elements, that work by iteratively recognizing patterns in each other and then embodying these patterns as new system elements. This is seen as distinguishing mind from "self-organization" in general, as the latter is not so focused on continual pattern recognition. Colloquially this means that "a mind is a system continually creating itself via recognizing patterns in itself."
- **Combo**: The programming language used internally by MOSES to represent the programs it evolves. SchemaNodes may refer to Combo programs, whether the latter are learned via MOSES or via some other means. The textual realization of Combo resembles LISP with less syntactic sugar. Internally a Combo program is represented as a program tree.
- **Composer**: In the PLN design, a rule is denoted a composer if it needs premises for generating its consequent. See generator.

- **CogBuntu**: an Ubuntu Linux remix that contains all required packages and tools to test and develop OpenCog.
- **Concept Creation**: A general term for cognitive processes that create new ConceptNodes, PredicateNodes or concept maps representing new concepts.
- **Conceptual Blending**: A process of creating new concepts via judiciously combining pieces of old concepts. This may occur in OpenCog in many ways, among them the explicit use of a ConceptBlending MindAgent, that blends two or more ConceptNodes into a new one.
- **Confidence**: A component of an OpenCog/PLN TruthValue, which is a scaling into the interval [0,1] of the weight of evidence associated with a truth value. In the simplest case (of a probabilistic Simple Truth Value), one uses confidence c = n / (n+k), where n is the weight of evidence and k is a parameter. In the case of an Indefinite Truth Value, the confidence is associated with the width of the probability interval.
- **Confidence Decay**: The process by which the confidence of an Atom decreases over time, as the observations on which the Atom's truth value is based become increasingly obsolete. This may be carried out by a special MindAgent. The rate of confidence decay is subtle and contextually determined, and must be estimated via inference rather than simply assumed a priori.
- **Consciousness**: CogPrime is not predicated on any particular conceptual theory of consciousness. Informally, the AttentionalFocus is sometimes referred to as the "conscious" mind of a CogPrime system, with the rest of the Atomspace as "unconscious" Ð but this is just an informal usage, not intended to tie the CogPrime design to any particular theory of consciousness. The primary originator of the CogPrime design (Ben Goertzel) tends toward panpsychism, as it happens.
- **Context**: In addition to its general common-sensical meaning, in CogPrime the term Context also refers to an Atom that is used as the first argument of a ContextLink. The second argument of the ContextLink then contains Links or Nodes, with TruthValues calculated only restricted to the context defined by the first argument. For instance, (ContextLink USA (InheritanceLink person obese)).
- **Core**: The MindOS portion of OpenCog, comprising the Atomspace, the CogServer, and other associated "infrastructural" code.
- **Corrective Learning**: When an agent learns how to do something, by having another agent explicitly guide it in doing the thing. For instance, teaching a dog to sit by pushing its butt to the ground.
- **CSDLN**: Compositional Spatiotemporal Deep Learning Network): A hierarchical pattern recognition network, in which each layer corresponds to a certain spatiotemporal granularity, the nodes on a given layer correspond to spatiotemporal regions of a given size, and the children of a node correspond to sub-regions of the region the parent corresponds to.

Jeff Hawkins HTM is one example CSDLN, and Itamar Arel's DeSTIN (currently used in OpenCog) is another.

- **Declarative knowledge**: Semantic knowledge as would be expressed in propositional or predicate logic Ð facts or beliefs.
- **Deduction**: In general, this refers to the derivation of conclusions from premises using logical rules. In PLN in particular, this often refers to the exercise of a specific inference rule, the PLN Deduction rule (A $\rightarrow$ B, B $\rightarrow$ C, therefore A$\rightarrow$ C)
- **Deep Learning**: Learning in a network of elements with multiple layers, involving feedforward and feedback dynamics, and adaptation of the links between the elements. An example deep learning algorithm is DeSTIN, which is being integrated with OpenCog for perception processing.
- **Defrosting**: Restoring, into the RAM portion of an Atomspace, an Atom (or set thereof) previously saved to disk.
- **Demand**: In CogPrime 's OpenPsi subsystem, this term is used in a manner inherited from the Psi model of motivated action. A Demand in this context is a quantity whose value the system is motivated to adjust. Typically the system wants to keep the Demand between certain minimum and maximum values. An Urge develops when a Demand deviates from its target range.
- **Deme**: In MOSES, an "island" of candidate programs, closely clustered together in program space, being evolved in an attempt to optimize a certain fitness function. The idea is that within a deme, programs are generally similar enough that reasonable syntax-semantics correlation obtains.
- **Derived Hypergraph**: The SMEPH hypergraph obtained via modeling a system in terms of a hypergraph representing its internal states and their relationships. For instance, a SMEPH vertex represents a collection of internal states that habitually occur in relation to similar external situations. A SMEPH edge represents a relationship between two SMEPH vertices (e.g. a similarity or inheritance relationship). The terminology "edge /vertex" is used in this context, to distinguish from the "link / node" terminology used in the context of the Atomspace.
- **DeSTIN**: A specific CSDLN created by Itamar Arel, tested on visual perception, and appropriate for integration within CogPrime .
- **Dialogue**: Linguistic interaction between two or more parties. In a CogPrime
  context, this may be in English or another natural language, or it may be in Lojban or Psynese.
- **Dialogue Control**: The process of determining what to say at each juncture in a dialogue. This is distinguished from the linguistic aspects of dialogue, language comprehension and language generation. Dialogue control applies to Psynese or Lojban, as well as to human natural language.
- **Dimensional Embedding**: The process of embedding entities from some non-dimensional space (e.g. the Atomspace) into an n-dimensional

Euclidean space. This can be useful in an AI context because some sorts of queries (e.g. "find everything similar to X", "find a path between X and Y") are much faster to carry out among points in a Euclidean space, than among entities in a space with less geometric structure.

- **Distributed Atomspace**: An implementation of an Atomspace that spans multiple computational processes; generally this is done to enable spreading an Atomspace across multiple machines.
- **Dual Network**: A network of mental or informational entities with both a hierarchical structure and a heterarchical structure, and an alignment among the two structures so that each one helps with the maintenance of the other. This is hypothesized to be a critical emergent structure, that must emerge in a mind (e.g. in an Atomspace) in order for it to achieve a reasonable level of human-like general intelligence (and possibly to achieve a high level of pragmatic general intelligence in any physical environment).
- **Efficient Pragmatic General Intelligence**: A formal, mathematical definition of general intelligence (extending the pragmatic general intelligence), that ultimately boils down to: the ability to achieve complex goals in complex environments using limited computational resources (where there is a specifically given weighting function determining which goals and environments have highest priority). More specifically, the definition weighted-sums the system's normalized goal-achieving ability over (goal, environment) pairs, where the weights are given by some assumed measure over (goal, environment pairs), and where the normalization is done via dividing by the (space and time) computational resources used for achieving the goal.
- **Elegant Normal Form (ENF)**: Used in MOSES, this is a way of putting programs in a normal form while retaining their hierarchical structure. This is critical if one wishes to probabilistically model the structure of a collection of programs, which is a meaningful operation if the collection of programs is operating within a region of program space where syntax-semantics correlation holds to a reasonable degree. The Reduct library is used to place programs into ENF.
- **Embodied Communication Prior**: The class of prior distributions over goal/environment pairs, that are imposed by placing an intelligent system in an environment where most of its tasks involve controlling a spatially localized body in a complex world, and interacting with other intelligent spatially localized bodies. It is hypothesized that many key aspects of human-like intelligence (e.g. the use of different subsystems for different memory types, and cognitive synergy between the dynamics associated with these subsystems) are consequences of this prior assumption. This is related to the Mind-World Correspondence Principle.
- **Embodiment**: Colloquially, in an OpenCog context, this usually means the use of an AI software system to control a spatially localized body in a complex (usually 3D) world. There are also possible "borderline cases"

of embodiment, such as a search agent on the Internet. In a sense any AI is embodied, because it occupies some physical system (e.g. computer hardware) and has some way of interfacing with the outside world.

- **Emergence**: A property or pattern in a system is emergent if it arises via the combination of other system components or aspects, in such a way that its details would be very difficult (not necessarily impossible in principle) to predict from these other system components or aspects.
- **Emotion**: Emotions are system-wide responses to the system's current and predicted state. Dorner's Psi theory of emotion contains explanations of many human emotions in terms of underlying dynamics and motivations, and most of these explanations make sense in a CogPrime context, due to CogPrime 's use of OpenPsi (modeled on Psi) for motivation and action selection.
- **Episodic Knowledge**: Knowledge about episodes in an agent's life history, or the life-history of other agents. CogPrime includes a special dimensional embedding space only for episodic knowledge, easing organization and recall.
- **Evolutionary Learning**: Learning that proceeds via the rough process of iterated differential reproduction based on fitness, incorporating variation of reproduced entities. MOSES is an explicitly evolutionary-learning-based portion of CogPrime ; but CogPrime 's dynamics as a whole may also be conceived as evolutionary.
- **Exemplar**: (in the context of imitation learning) - When the owner wants to teach an OpenCog controlled agent a behavior by imitation, he/she gives the pet an exemplar. To teach a virtual pet "fetch" for instance, the owner is going to throw a stick, run to it, grab it with his/her mouth and come back to its initial position.
- **Exemplar**: (in the context of MOSES) – Candidate chosen as the core of a new deme , or as the central program within a deme, to be varied by representation building for ongoing exploration of program space
- **Explicit Knowledge Representation**: Knowledge representation in which individual, easily humanly identifiable pieces of knowledge correspond to individual elements in a knowledge store (elements that are explicitly there in the software and accessible via very rapid, deterministic operations)
- **Extension**: In PLN, the extension of a node refers to the instances of the category that the node represents. In contrast is the intension.
- **Fishgram (Frequent and Interesting Sub-hypergraph Mining)**: A pattern mining algorithm for identifying frequent and/or interesting sub-hypergraphs in the Atomspace.
- **First-Order Inference (FOI)**: The subset of PLN that handles Logical Links not involving VariableAtoms or higher-order functions. The other aspect of PLN, Higher-Order Inference, uses Truth Value formulas derives from First-Order Inference.

- **Forgetting**: The process of removing Atoms from the in-RAM portion of Atomspace, when RAM gets short and they are judged not as valuable to retain in RAM as other Atoms. This is commonly done using the LTI values of the Atoms (removing lowest LTI-Atoms, or more complex strategies involving the LTI of groups of interconnected Atoms). May be done by a dedicated Forgetting MindAgent. VLTI may be used to determine the fate of forgotten Atoms.
- **Forward Chainer**: A control mechanism (MindAgent) for PLN inference, that works by taking existing Atoms and deriving conclusion from them using PLN rules, and then iterating this process. The goal is to derive new Atoms that are interesting according to some given criterion.
- **Frame2Atom**: A simple system of hand-coded rules for translating the output of RelEx2Frame (logical representation of semantic relationships using FrameNet relationships) into Atoms
- **Freezing**: Saving Atoms from the in-RAM Atomspace to disk
- **General Intelligence**: Often used in an informal, commonsensical sense, to mean the ability to learn and generalize beyond specific problems or contexts. Has been formalized in various ways as well, including formalizations of the notion of "achieving complex goals in complex environments" and "achieving complex goals in complex environments using limited resources." Usually interpreted as a fuzzy concept, according to which absolutely general intelligence is physically unachievable, and humans have a significant level of general intelligence, but far from the maximally physically achievable degree.
- **Generalized Hypergraph**: A hypergraph with some additional features, such as links that point to links, and nodes that are seen as "containing" whole sub-hypergraphs. This is the most natural and direct way to mathematically/visually model the Atomspace.
- **Generator**: In the PLN design, a rule is denoted a generator if it can produce its consequent without needing premises (e.g. LookupRule, which just looks it up in the AtomSpace). See composer.
- **Global, Distributed Memory**: Memory that stores items as implicit knowledge, with each memory item spread across multiple components, stored as a pattern of organization or activity among them.
- **Glocal Memory**: The storage of items in memory in a way that involves both localized and global, distributed aspects.
- **Goal**: An Atom representing a function that a system (like OpenCog) is supposed to spend a certain non-trivial percentage of its attention optimizing. The goal, informally speaking, is to maximize the Atom's truth value.
- **Goal, Implicit**: A goal that an intelligent system, in practice, strives to achieve; but that is not explicitly represented as a goal in the system's knowledge base.
- **Goal, Explicit**: A goal that an intelligent system explicitly represents in its knowledge base, and expends some resources trying to achieve. Goal

Nodes (which may be Nodes or, e.g. ImplicationLinks) are used for this purpose in OpenCog.

- **Goal-Driven Learning**: Learning that is driven by the cognitive schematic Ð i.e. by the quest of figuring out which procedures can be expected to achieve a certain goal in a certain sort of context.
- **Grounded SchemaNode**: See SchemaNode, Grounded
- **Hebbian Learning**: An aspect of Attention Allocation, centered on creating and updating HebbianLinks, which represent the simultaneous importance of the Atoms joined by the HebbianLink
- **Hebbian Links**: Links recording information about the associative relationship (co-occurrence) between Atoms. These include symmetric and asymmetric HebbianLinks.
- **Heterarchical Network**: A network of linked elements in which the semantic relationships associated with the links are generally symmetrical (e.g. they may be similarity links, or symmetrical associative links). This is one important sort of subnetwork of an intelligent system; see Dual Network.
- **Hierarchical Network**: A network of linked elements in which the semantic relationships associated with the links are generally asymmetrical, and the parent nodes of a node have a more general scope and some measure of control over their children (though there may be important feedback dynamics too). This is one important sort of subnetwork of an intelligent system; see Dual Network.
- **Higher-Order Inference (HOI)**: PLN inference involving variables or higher-order functions. In contrast to First-Order Inference (FOI).
- **Hillclimbing**: A general term for greedy, local optimization techniques, including some relatively sophisticated ones that involve "mildly nonlocal" jumps.
- **Human-Level Intelligence**: General intelligence that's "as smart as" human general intelligence, even if in some respects quite unlike human intelligence. An informal concept, which generally doesn't come up much in CogPrime work, but is used frequently by some other AI theorists.
- **Human-Like Intelligence**: General intelligence with properties and capabilities broadly resembling those of humans, but not necessarily precisely imitating human beings.
- **Hypergraph**: A conventional hypergraph is a collection of nodes and links, where each link may span any number of nodes. OpenCog makes use of generalized hypergraphs (the Atomspace is one of these).
- **Imitation Learning**: Learning via copying what some other agent is observed to do.
- **Implication**: Often refers to an ImplicationLink between two PredicateNodes, indicating an (extensional, intensional or mixed) logical implication.
- **Implicit Knowledge Representation**: Representation of knowledge via having easily humanly identifiable pieces of knowledge correspond to

the pattern of organization and/or dynamics of elements, rather than via having individual elements correspond to easily humanly identifiable pieces of knowledge.

- **Importance**: A generic term for the Attention Values associated with Atoms. Most commonly these are STI (short term importance) and LTI (long term importance) values. Other importance values corresponding to various different time scales are also possible. In general an importance value reflects an estimate of the likelihood an Atom will be useful to the system over some particular future time-horizon. STI is generally relevant to processor time allocation, whereas LTI is generally relevant to memory allocation.
- **Importance Decay**: The process of Atom' importance values (e.g. STI and LTI) decreasing over time, if the Atoms are not utilized. Importance decay rates may in general be context-dependent.
- **Importance Spreading**: A synonym for Importance Updating, intended to highlight the similarity with "activation spreading" in neural and semantic networks
- **Importance Updating**: The CIM-Dynamic that periodically (frequently) updates the STI and LTI values of Atoms based on their recent activity and their relationships.
- **Imprecise Truth Value**: Peter Walley's imprecise truth values are intervals [L,U], interpreted as lower and upper bounds of the means of probability distributions in an envelope of distributions. In general, the term may be used to refer to any truth value involving intervals or related constructs, such as indefinite probabilities.
- **Indefinite Probability**: An extension of a standard imprecise probability, comprising a credible interval for the means of probability distributions governed by a given second-order distribution.
- **Indefinite Truth Value**: An OpenCog TruthValue object wrapping up an indefinite probability
- **Induction**: In PLN, a specific inference rule (A $\rightarrow$ B, A $\rightarrow$ C, therefore B -$\rightarrow$ C). In general, the process of heuristically inferring that what has been seen in multiple examples, will be seen again in new examples. Induction in the broad sense, may be carried out in OpenCog by methods other than PLN induction. When emphasis needs to be laid on the particular PLN inference rule, the phrase "PLN Induction" is used.
- **Inference**: Generally speaking, the process of deriving conclusions from assumptions. In an OpenCog context, this often refers to the PLN inference system. Inference in the broad sense is distinguished from general learning via some specific characteristics, such as the intrinsically incremental nature of inference: it proceeds step by step.
- **Inference Control**: A cognitive process that determines what logical inference rule (e.g. what PLN rule) is applied to what data, at each point in the dynamic operation of an inference process

- **Integrative AGI**: An AGI architecture, like CogPrime , that relies on a number of different powerful, reasonably general algorithms all cooperating together. This is different from an AGI architecture that is centered on a single algorithm, and also different than an AGI architecture that expects intelligent behavior to emerge from the collective interoperation of a number of simple elements (without any sophisticated algorithms coordinating their overall behavior).
- **Integrative Cognitive Architecture**: A cognitive architecture intended to support integrative AGI.
- **Intelligence**: An informal, natural language concept. "General intelligence" is one slightly more precise specification of a related concept; "Universal intelligence" is a fully precise specification of a related concept. Other specifications of related concepts made in the particular context of CogPrime research are the pragmatic general intelligence and the efficient pragmatic general intelligence.
- **Intension**: In PLN, the intention of a node consists of Atoms representing properties of the entity the node represents
- **Intentional memory**: A system's knowledge of its goals and their subgoals, and associations between these goals and procedures and contexts (e.g. cognitive schematics).
- **Internal Simulation World**: A simulation engine used to simulate an external environment (which may be physical or virtual), used by an AGI system as its "mind's eye" in order to experiment with various action‘ q sequences and envision their consequences, or observe the consequences of various hypothetical situations. Particularly important for dealing with episodic knowledge.
- **Interval Algebra**: Allen Interval Algebra, a mathematical theory of the relationships between time intervals. CogPrime utilizes a fuzzified version of classic Interval Algebra.
- **IRC Learning (Imitation, Reinforcement, Correction)**: Learning via interaction with a teacher, involving a combination of imitating the teacher, getting explicit reinforcement signals from the teacher, and having one's incorrect or suboptimal behaviors guided toward betterness by the teacher in real-time. This is a large part of how young humans learn.
- **Knowledge Base**: A shorthand for the totality of knowledge possessed by an intelligent system during a certain interval of time (whether or not this knowledge is explicitly represented). Put differently: this is an intelligence's total memory contents (inclusive of all types of memory) during an interval of time.
- **Language Comprehension**: The process of mapping natural language speech or text into a more "cognitive", largely language-independent representation. In OpenCog this has been done by various pipelines consisting of dedicated natural language processing tools, e.g. a pipeline: text → Link Parser → RelEx → RelEx2Frame → Frame2Atom Ð Atomspace; and alternatively a pipeline Link Parser → Link2Atom → Atomspace.

It would also be possible to do language comprehension purely via PLN and other generic OpenCog processes, without using specialized language processing tools.

- **Language Generation**: The process of mapping (largely language-independent) cognitive content into speech or text. In OpenCog this has been done by various pipelines consisting of dedicated natural language processing tools, e.g. a pipeline: Atomspace → NLGen → text; or more recently Atomspace → Atom2Link → surface realization → text. It would also be possible to do language generation purely via PLN and other generic OpenCog processes, without using specialized language processing tools.
- **Language Processing**: Processing of human language is decomposed, in CogPrime , into Language Comprehension, Language Generation, and Dialogue Control
- **Learning**: In general, the process of a system adapting based on experience, in a way that increases its intelligence (its ability to achieve its goals). The theory underlying CogPrime doesn't distinguish learning from reasoning, associating, or other aspects of intelligence.
- **Learning Server**: In some OpenCog configurations, this refers to a software server that performs "offline" learning tasks (e.g. using MOSES or hillclimbing), and is in communication with an Operational Agent Controller software server that performs real-time agent control and dispatches learning tasks to and receives results from the Learning Server.
- **Linguistic Links**: A catch-all term for Atoms explicitly representing linguistic content, e.g. WordNode, SentenceNode, CharacterNodeÉ.
- **Link**: A type of Atom, representing a relationship among one or more Atoms. Links and Nodes are the two basic kinds of Atoms.
- **Link Parser**: A natural language syntax parser, created by Sleator and Temperley at Carnegie-Mellon University, and currently used as part of OpenCogPrime
  s natural language comprehension and natural language generation system.
- **Link2Atom**: A system for translating link parser links into Atoms. It attempts to resolve precisely as much ambiguity as needed in order to translate a given assemblage of link parser links into a unique Atom structure.
- **Lobe**: A term sometime used to refer to a portion of a distributed Atomspace that lives in a single computational process. Often different lobes will live on different machines.
- **Localized Memory**: Memory that stores each item using a small number of closely-connected elements.
- **Logic**: In an OpenCog context, this usually refers to a set of formal rules for translating certain combinations of Atoms into "conclusion" Atoms. The paradigm case at present is the PLN probabilistic logic system, but OpenCog can also be used together with other logics.

- **Logical Links**: Any Atoms whose truth values are primarily determined or adjusted via logical rules, e.g. PLN's InheritanceLink, SimilarityLink, ImplicationLink, etc. The term isn't usually applied to other links like HebbianLinks whose semantics isn't primarily logic-based, even though these other links can be processed via (e.g. PLN) logical inference via interpreting them logically.
- **Lojban**: A constructed human language, with a completely formalized syntax and a highly formalized semantics, and a small but active community of speakers. In principle this seems an extremely good method for communication between humans and early-stage AGI systems.
- **Lojban++**: A variant of Lojban that incorporates English words, enabling more flexible expression without the need for frequent invention of new Lojban words.
- **Long Term Importance (LTI)**: A value associated with each Atom, indicating roughly the expected utility to the system of keeping that Atom in RAM rather than saving it to disk or deleting it. It's possible to have multple LTI values pertaining to different time scales, but so far practical implementation and most theory has centered on the option of a single LTI value.
- **LTI**: Long Term Importance
- **Map**: A collection of Atoms that are interconnected in such a way that they tend to be commonly active (i.e. to have high STI, e.g. enough to be in the AttentionalFocus, at the same time)
- **Map Encapsulation**: The process of automatically identifying maps in the Atomspace, and creating Atoms that "encapsulate" them; the Atom encapsulation a map would link to all the Atoms in the map. This is a way of making global memory into local memory, thus making the system's memory glocal and explicitly manifesting the "cognitive equation." This may be carried out via a dedicated MapEncapsulation MindAgent.
- **Map Formation**: The process via which maps form in the Atomspace. This need not be explicit; maps may form implicitly via the action of Hebbian Learning. It will commonly occur that Atoms frequently co-occurring in the AttentionalFocus, will come to be joined together in a map.
- **Memory types**: In CogPrime
  this generally refers to the different types of memory that are embodied in different data structures or processes in the CogPrime
  architecture, e.g. declarative (semantic), procedural, attentional, intentional, episodic, sensorimotor.
- **Mind-World Correspondence Principle**: The principle that, for a mind to display efficient pragmatic general intelligence relative to a world, it should display many of the same key structural properties as that world. This can be formalized by modeling the world as mind as probabilistic state transition graphs, and saying that the categories implicit in the

state transition graphs of the mind and world should be inter-mappable via a high-probability morphism.

- **Mind OS**: A synonym for the OpenCog Core
- **MindAgent**: An OpenCog software object, residing in the CogServer, that carries out some processes in interaction with the Atomspace. A given conceptual cognitive process (e.g. PLN inference, Attention allocation, etc.) may be carried out by a number of different MindAgents designed to work together.
- **Mindspace**: A model of the set of states of an intelligent system as a geometrical space, imposed by assuming some metric on the set of mindstates. This may be used as a tool for formulating general principles about the dynamics of generally intelligent systems.
- **Modulators**: Parameters in the Psi model of motivated, emotional cognition, that modulate the way a system perceives, reasons about and interacts with the world.
- **MOSES (Meta-Optimizing Semantic Evolutionary Search)**: An algorithm for procedure learning, which in the current implementation learns programs in the Combo language. MOSES is an evolutionary learning system, which differs from typical genetic programming systems in multiple aspects including: a subtler framework for managing multiple "demes" or "islands" of candidate programs; a library called Reduct for placing programs in Elegant Normal Form; and the use of probabilistic modeling in place of, or in addition to, mutation and crossover as means of determining which new candidate programs to try.
- **Motoric**: Pertaining to the control of physical actuators, e.g. those connected to a robot. May sometimes be used to refer to the control of movements of a virtual character as well.
- **Moving Bubble of Attention**: The Attentional Focus of a CogPrime system
- **Natural Language Comprehension**: See Language Comprehension
- **Natural Language Generation**: See Language Generation
- **Natural Language Processing**: See Language Processing
- **NLGen**: Software for carrying out the surface realization phase of natural language generation, via translating collections of RelEx output relationships into English sentences. Was made functional for simple sentences and some complex sentences; not currently under active development, as work has shifted to the related Atom2Link approach to language generation.
- **Node**: A type of Atom. Links and Nodes are the two basic kinds of Atoms. Nodes, mathematically, can be thought of as "0-ary links. Some types Nodes refer to external or mathematical entities (e.g. WordNode, NumberNode); others are purely abstract, e.g. a ConceptNode is characterized purely by the Links relating it to other atoms. GroundedPredicateNodes and GroundedSchemaNodes connect to explicitly represented procedures (sometimes in the Combo language); ungrounded PredicateN-

odes and SchemaNodes are abstract and, like ConceptNodes, purely characterized by their relationships.

- **Node Probability**: Many PLN inference rules rely on probabilities associated with Nodes. Node probabilities are often easiest to interpret in a specific context, e.g. the probability P(cat) makes obvious sense in the context of a typical American house, or in the context of the center of the sun. Without any contextual specification, P(A) is taken to mean the probability that a randomly chosen occasion of the system's experience includes some instance of A.
- **Novamente Cognition Engine (NCE)**: A proprietary proto-AGI software system, the predecessor to OpenCog. Many parts of the NCE were open-sourced to form portions of OpenCog, but some NCE code was not included in OpenCog; and now OpenCog includes multiple aspects and plenty of code that was not in NCE.
- **OpenCog**: A software framework intended for development of AGI systems, and also for narrow-AI application using tools that have AGI applications. Co-designed with the CogPrime cognitive architecture, but not exclusively bound to it.
- **OpenCog Prime**: The implementation of the CogPrime cognitive architecture within the OpenCog software framework
- **OpenPsi**: CogPrime 's architecture for motivation-driven action selection, which is based on adapting Dorner's Psi model for use in the OpenCog framework.
- **Operational Agent Controller (OAC)**: In some OpenCog configurations, this is a software server containing a CogServer devoted to real-time control of an agent (e.g. a virtual world agent, or a robot). Background, offline learning tasks may then be dispatched to other software processes, e.g. to a Learning Server.
- **Pattern**: In a CogPrime context, the term "pattern" is generally used to refer to a process that produces some entity, and is judged simpler than that entity.
- **Pattern Mining**: Pattern mining is the process of extracting an (often large) number of patterns from some body of information, subject to some criterion regarding which patterns are of interest. Often (but not exclusively) it refers to algorithms that are rapid or "greedy", finding a large number of simple patterns relatively inexpensively.
- **Pattern Recognition**: The process of identifying and representing a pattern in some substrate (e.g. some collection of Atoms, or some raw perceptual data, etc.)
- **Patternism**: The philosophical principle holding that, from the perspective of engineering intelligent systems, it is sufficient and useful to think about mental processes in terms of (static and dynamical) patterns.
- **Perception**: The process of understanding data from sensors. When natural language is ingested in textual format, this is generally not considered perceptual. Perception may be taken to encompass both pre-processing

that prepares sensory data for ingestion into the Atomspace, processing via specialized perception processing systems like DeSTIN that are connected to the Atomspace, and more cognitive-level process within the Atomspace that is oriented toward understanding what has been sensed.

- **Piagetan Stages**: A series of stages of cognitive development hypothesized by developmental psychologist Jean Piaget, which are easy to interpret in the context of developing CogPrime systems. The basic stages are: Infantile, Pre-operational, Concrete Operational and Formal. Post-formal stages have been discussed by theorists since Piaget and seem relevant to AGI, especially advanced AGI systems capable of strong self-modification.
- **PLN**: short for Probabilistic Logic Networks
- **PLN, First-Order**: See First-Order Inference
- **PLN, Higher-Order**: See Higher-Order Inference
- **PLN Rules**: A PLN Rule takes as input one or more Atoms (the "premises", usually Links), and output an Atom that is a "logical conclusion" of those Atoms. The truth value of the consequence is determined by a PLN Formula associated with the Rule.
- **PLN Formulas**: A PLN Formula, corresponding to a PLN Rule, takes the TruthValues corresponding to the premises and produces the TruthValue corresponding to the conclusion. A single Rule may correspond to multiple Formulas, where each Formula deals with a different sort of TruthValue.
- **Pragmatic General Intelligence**: A formalization of the concept of general intelligence, based on the concept that general intelligence is the capability to achieve goals in environments, calculated as a weighted average over some fuzzy set of goals and environments.
- **Predicate Evaluation**: The process of determining the Truth Value of a predicate, embodied in a PredicateNode. This may be recursive, as the predicate referenced internally by a Grounded PredicateNode (and represented via a Combo program tree) may itself internally reference other PredicateNodes.
- **Probabilistic Logic Networks (PLN)**: A mathematical and conceptual framework for reasoning under uncertainty, integrating aspects of predicate and term logic with extensions of imprecise probability theory. OpenCogPrime
  s central tool for symbolic reasoning.
- **Procedural Knowledge**: Knowledge regarding which series of actions (or action-combinations) are useful for an agent to undertake in which circumstances. In CogPrime these may be learned in a number of ways, e.g. via PLN or via Hebbian learning of Schema Maps, or via explicit learning of Combo programs via MOSES or hillclimbing. Procedures are represented as SchemaNodes or Schema Maps.
- **Procedure Evaluation/Execution**: A general term encompassing both Schema Execution and Predicate Evaluation, both of which are similar

computational processes involving manipulation of Combo trees associated with ProcedureNodes.

- **Procedure Learning**: Learning of procedural knowledge, based on any method, e.g. evolutionary learning (e.g. MOSES), inference (e.g. PLN), reinforcement learning (e.g. Hebbian learning)
- **Procedure Node**: A SchemaNode or PredicateNode
- **Psi**: A model of motivated action, and emotion, originated by Dietrich Dorner and further developed by Joscha Bach, who incorporated it in his proto-AGI system MicroPsi. OpenCogPrime
  s motivated-action component, OpenPsi, is roughly based on the Psi model.
- **Psynese**: A system enabling different OpenCog instances to communicate without using natural language, via directly exchanging Atom subgraphs, using a special system to map references in the speaker's mind into matching references in the listener's mind.
- **Psynet Model**: An early version of the theory of mind underlying CogPrime , referred to in some early writings on the Webmind AI Engine and Novamente Cognition Engine. The concepts underlying the psyynet model are still part of the theory underlying CogPrime , but the name has been deprecated as it never really caught on.
- **Reasoning**: See inference
- **Reduct**: A code library, used within MOSES, applying a collection of hand-coded rewrite rules that transform Combo programs into Elegant Normal Form.
- **Region Connection Calculus**: A mathematical formalism describing a system of basic operations among spatial regions. Used in CogPrime as part of spatial inference, to provide relations and rules to be referenced via PLN and potentially other subsystems.
- **Reinforcement Learning**: Learning procedures via experience, in a manner explicitly guided to cause the learning of procedures that will maximize the system's expected future reward. CogPrime does this implicitly whenever it tries to learn procedures that will maximize some Goal whose Truth Value is estimated via an expected reward calculation (where "reward" may mean simply the Truth Value of some Atom defined as "reward"). Goal-driven learning is more general than reinforcement learning as thus defined; and the learning that CogPrime does, which is only partially goal-driven, is yet more general.
- **RelEx**: A software system used in OpenCog as part of natural language comprehension, to map the output of the link parser into more abstract semantic relationships. These more abstract relationships may then be entered directly into the Atomspace, or they may be further abstracted before being entered into the Atomspace, e.g. by RelEx2Frame rules.
- **RelEx2Frame**: A system of rules for translating RelEx output into Atoms, based on the FrameNet ontology. The output of the RelEx2Frame rules make use of the FrameNet library of semantic relationships. The cur-

rent (2012) RelEx2Frame rule-based is problematic and the RelEx2Frame system is deprecated as a result, in favor of Link2Atom. However, the ideas embodied in these rules may be useful; if cleaned up the rules might profitably be ported into the Atomspace as ImplicationLinks.

- **Representation Building**: A stage within MOSES, wherein a candidate Combo program tree (within a deme) is modified by replacing one or more tree nodes with alternative tree nodes, thus obtaining a new, different candidate program within that deme. This process currently relies on hand-coded knowledge regarding which types of tree nodes a given tree node should be experimentally replaced with (e.g. an AND node might sensibly be replaced with an OR node, but not so sensibly replaced with a node representing a "kick" action).
- **Request for Services (RFS)**: In CogPrime 's Goal-driven action system, a RFS is a package sent from a Goal Atom to another Atom, offering it a certain amount of STI currency if it is able to deliver the goal what it wants (an increase in its Truth Value). RFS's may be passed on, e.g. from goals to subgoals to sub-subgoals, but eventually an RFS reaches a Grounded SchemaNode, and when the corresponding Schema is executed, the payment implicit in the RFS is made.
- **Robot Preschool**: An AGI Preschool in our physical world, intended for robotically embodied AGIs
- **Robotic Embodiment**: Using an AGI to control a robot. The AGI may be running on hardware physically contained in the robot, or may run elsewhere and control the robot via networking methods such as wifi.
- **Scheduler**: Part of the CogServer that controls which processes (e.g. which MindAgents) get processor time, at which point in time.
- **Schema**: A "script" describing a process to be carried out. This may be explicit, as in the case of a GroundedSchemaNode, or implicit, as the case in Schema maps or ungrounded SchemaNodes.
- **Schema Encapsulation**: The process of automatically recognizing a Schema Map in an Atomspace, and creating a Combo (or other) program embodying the process carried out by this Schema Map, and then storing this program in the Procedure Repository and associating it with a particular SchemaNode. This translates distributed, global procedural memory into localized procedural memory. It's a special case of Map Encapsulation.
- **Schema Execution**: The process of "running" a Grounded Schema, similar to running a computer program. Or, phrased alternately: The process of executing the Schema referenced by a Grounded SchemaNode. This may be recursive, as the predicate referenced internally by a Grounded SchemaNode (and represented via a Combo program tree) may itself internally reference other Grounded SchemaNodes.
- **Schema, Grounded**: A Schema that is associated with a specific executable program (either a Combo program or, say, C++ code)

- **Schema Map**: A collection of Atoms, including SchemaNodes, that tend to be enacted in a certain order (or set of orders), thus habitually enacting the same process. This is a distributed, globalized way of storing and enacting procedures.
- **Schema, Ungrounded**: A Schema that represents an abstract procedure, not associated with any particular executable program.
- **Schematic Implication**: A general, conceptual name for implications of the form ((Context AND Procedure) IMPLIES Goal)
- **SegSim**: A name for the main algorithm underlying the NLGen language generation software. The algorithm is based on segmenting a collection of Atoms into small parts, and matching each part against memory to find, for each part, cases where similar Atom-collections already have known linguistic expression.
- **Self-Modification**: A term generally used for AI systems that can purposefully modify their core algorithms and representations. Formally and crisply distinguishing this sort of "strong self-modification" from "mere" learning is a tricky matter.
- **Sensorimotor**: Pertaining to sensory data, motoric actions, and their combination and intersection.
- **Sensory**: Pertaining to data received by the AGI system from the outside world. In a CogPrime system that perceives language directly as text, the textual input will generally not be considered as "sensory" (on the other hand, speech audio data would be considered as "sensory").
- **Short Term Importance**: A value associated with each Atom, indicating roughly the expected utility to the system of keeping that Atom in RAM rather than saving it to disk or deleting it. It's possible to have multple LTI values pertaining to different time scales, but so far practical implementation and most theory has centered on the option of a single LTI value.
- **Similarity**: a link type indicating the probabilistic similarity between two different Atoms. Generically this is a combination of Intensional Similarity (similarity of properties) and Extensional Similarity (similarity of members).
- **Simple Truth Value**: a TruthValue involving a pair (s,d) indicating strength (e.g. probability or fuzzy set membership) and confidence d. d may be replaced by other options such as a count n or a weight of evidence w.
- **Simulation World**: See Internal Simulation World
- **SMEPH (Self-Modifying Evolving Probabilistic Hypergraphs)**: a style of modeling systems, in which each system is associated with a derived hypergraph
- **SMEPH Edge**: A link in a SMEPH derived hypergraph, indicating an empirically observed relationship (e.g. inheritance or similarity) between two

- **SMEPH Vertex**: A node in a SMEPH derived hypergraph representing a system, indicating a collection of system states empirically observed to arise in conjunction with the same external stimuli
- **Spatial Inference**: PLN reasoning including Atoms that explicitly reference spatial relationships
- **Spatiotemporal Inference**: PLN reasoning including Atoms that explicitly reference spatial and temporal relationships
- **STI**: Shorthand for Short Term Importance
- **Strength**: The main component of a TruthValue object, lying in the interval [0,1], referring either to a probability (in cases like InheritanceLink, SimilarityLink, EquivalenceLink, ImplicationLink, etc.) or a fuzzy value (as in MemberLink, EvaluationLink).
- **Strong Self-Modification**: This is generally used as synonymous with Self-Modification, in a CogPrime context.
- **Subsymbolic**: Involving processing of data using elements that have no correspondence to natural language terms, nor abstract concepts; and that are not naturally interpreted as symbolically "standing for" other things. Often used to refer to processes such as perception processing or motor control, which are concerned with entities like pixels or commands like "rotate servomotor 15 by 10 degrees theta and 55 degrees phi." The distinction between "symbolic" and "subsymbolic" is conventional in the history of AI, but seems difficult to formalize rigorously. Logic-based AI systems are typically considered "symbolic", yet
- **Supercompilation**: A technique for program optimization, which globally rewrites a program into a usually very different looking program that does the same thing. A prototype supercompiler was applied to Combo programs with successful results.
- **Surface Realization**: The process of taking a collection of Atoms and transforming them into a series of words in a (usually natural) language. A stage in the overall process of language generation.
- **Symbol Grounding**: The mapping of a symbolic term into perceptual or motoric entities that help define the meaning of the symbolic term. For instance, the concept "Cat" may be grounded by images of cats, experiences of interactions with cats, imaginations of being a cat, etc.
- **Symbolic**: Pertaining to the formation or manipulation of symbols, i.e. mental entities that are explicitly constructed to represent other entities. Often contrasted with subsymbolic.
- **Syntax-Semantics Correlation**: In the context of MOSES and program learning more broadly, this refers to the property via which distance in syntactic space (distance between the syntactic structure of programs, e.g. if they're represented as program trees) and semantic space (distance between the behaviors of programs, e.g. if they're represented as sets of input/output pairs) are reasonably well correlated. This can often happen among sets of programs that are not too widely dispersed in program space. The Reduct library is used to place Combo programs

in Elegant Normal Form, which increases the level of syntax-semantics programs between them. The programs in a single MOSES deme are often closely enough clustered together that they have reasonably high syntax-semantics correlation.

- **System Activity Table**: An OpenCog component that records information regarding what a system did in the past.
- **Temporal Inference**: Reasoning that heavily involves Atoms representing temporal information, e.g. information about the duration of events, or their temporal relationship (before, after, during, beginning, ending). As implemented in CogPrime , makes use of an uncertain version of Allen Interval Algebra.
- **Truth Value**: A package of information associated with an Atom, indicating its degree of truth. SimpleTruthValue and IndefiniteTruthValue are two common, particular kinds. Multiple truth values associated with the same Atom from different perspectives may be grouped into CompositeTruthValue objects.
- **Universal Intelligence**: A technical term introduced by Shane Legg and Marcus Hutter, describing (roughly speaking) the average capability of a system to carry out computable goals in computable environments, where goal/environment pairs are weighted via the length of the shortest program for computing them.
- **Urge**: In OpenPsi, an Urge develops when a Demand deviates from its target range.
- **Very Long Term Importance (VLTI)**: A bit associated with Atoms, which determines whether, when an Atom is forgotten (removed from RAM), it is saved to disk (frozen) or simply deleted.
- **Virtual AGI Preschool**: A virtual world intended for AGI teaching/-training/learning, bearing broad resemblance to the preschool environments used for young humans.
- **Virtual Embodiment**: Using an AGI to control an agent living in a virtual world or game world, typically (but not necessarily) a 3D world with broad similarity to the everyday human world.
- **Webmind AI Engine**: A predecessor to the Novamente Cognition Engine and OpenCog, developed 1997-2001 Ð with many similar concepts (and also some different ones) but quite different algorithms and software architecture

Appendix B
Steps Toward a Formal Theory of Cognitive Structure and Dynamics

B.1 Introduction

Transforming the conceptual and formal ideas of Section **??** into rigorous mathematical theory will be a large enterprise, and is not something we have achieved so far. However, we do believe we have some idea regarding what kind of mathematical and conceptual toolset will be useful for enacting this transformation. In this appendix we will elaborate our ideas regarding this toolset, and in the process present some concrete notions such as a novel mathematical formulation of the concept of cognitive synergy, and a more formal statement of many of the "key claims" regarding CogPrime given in Chapter 6.

The key ideas involved here are: modeling multiple memory types as mathematical categories (with functors mapping between them), modeling memory items as probability distributions, and measuring distance between memory items using two metrics, one based on algorithmic information theory and one on classical information geometry. Building on these ideas, core hypotheses are then presented:

- a **syntax-semantics correlation** principle, stating that in a successful AGI system, these two metrics should be roughly correlated
- a **cognitive geometrodynamics** principle, stating that on the whole intelligent minds tend to follow geodesics (shortest paths) in mindspace, according to various appropriately defined metrics (e.g. the metric measuring the distance between two entities in terms of the length and/or runtime of the shortest programs computing one from the other).
- a **cognitive synergy** principle, stating that shorter paths may be found through the composite mindspace formed by considering multiple memory types together, than by following the geodesics in the mindspaces corresponding to individual memory types.

These ideas are not strictly necessary for understanding the CogPrime design as outlined in Part 2 of this book. However, our hope is that they will

be helpful later on for elaborating a deeper theoretical understanding of CogPrime , and hence in developing the technical aspects of the CogPrime design beyond the stage presented in Part 2. Our sense is that, ultimately, the theory and practice of AGI will both go most smoothly if they can proceed together, with theory guiding algorithm and architecture tuning, but also inspired by lessons learned via practical experimentation. At present the CogPrime design has been inspired by a combination of broad theoretical notions about the overall architecture, and specific theoretical calculations regarding specific components. One of our hopes is that in later versions of CogPrime , precise theoretical calculations regarding the overall architecture may also be possible, perhaps using ideas descending from those in this appendix.

B.2 Modeling Memory Types Using Category Theory

We begin by formalizing the different types of memory critical for a human-like integrative AGI system, in a manner that makes it easy to study mappings between different memory types. One way to do this is to consider each type of memory as a *category*, in the sense of category theory. Specifically, in this section we roughly indicate how one may model declarative, procedural, episodic, attentional and intentional categories, thus providing a framework in which mapping between these different memory types can be modeled using functors. The discussion is quite brief and general, avoiding commitments about how memories are implemented.

B.2.1 The Category of Procedural Memory

We model the space of procedures as a *graph*. We assume there exists a set T of "atomic transformations" on the category C_{Proc} of procedures, so that each $t \in T$ maps an input procedure into a unique output procedure. We then consider a labeled digraph whose nodes are objects in C_{Proc} (i.e. procedures), and which has a link labeled t between procedure P_1 and P_2 if t maps P_1 into P_2. Morphisms on program space may then be taken as paths in this digraph, i.e. as composite procedure transformations defined by sequences of atomic procedure transformations.

As an example, if procedures are represented as *ensembles of program trees*, where program trees are defined in the manner suggested in [LG09] and 21, then one can consider tree edit operations as defined in [Bil05] as one's atomic transformations. If procedures are represented as formal neural nets or ensembles thereof, one can take a similar approach.

B.2.2 The Category of Declarative Memory

The category C_{Dec} of declarative knowledge may be handled somewhat similarly, via assuming the existence of a set of transformations between declarative knowledge items, constructing a labeled digraph induced by these transformations, and defining morphisms as paths in this digraph. For example, if declarative knowledge items are represented as expressions in some logical language, then transformations may be naturally taken to correspond to inference steps in the associated logic system. Morphisms then represent sequences of inference steps that transform one logical expression into another.

B.2.3 The Category of Episodic Memory

What about episodic memory – the record an intelligence keeps of its own experiences? Given that we are talking about intelligences living in a world characterized by 3 spatial dimensions and one temporal dimension, one way to model a remembered episode (i.e., an object in the category C_{Ep} of episodic memories) is as as a scalar field defined over a grid-cell discretization of 4D spacetime. The scalar field, integrated over some region of spacetime, tells the extent to which that region belongs to the episode. In this way one may also consider episodes as fuzzy sets of spacetime regions. We may then consider a category whose objects are *episode-sets*, i.e. fuzzy sets of fuzzy sets of spacetime regions.

To define morphisms on the space of episode-sets, one approach is to associate an episode $\mathcal{E}$ with the set $\mathcal{P}_{\mathcal{E},\epsilon}$ of programs that calculate the episode within a given error ϵ. One may then construct a graph whose nodes are episode-sets, and in which $\mathcal{E}_1$ is linked to $\mathcal{E}_2$ if applying an atomic procedure-transformation to some program in $\mathcal{P}_{\mathcal{E}_1,\epsilon}$ yields a program in $\mathcal{P}_{\mathcal{E}_2,\epsilon}$.

B.2.4 The Category of Intentional Memory

To handle the category C_{Int} of intentional knowledge, we recall that in our formal agents model, goals are functions. Therefore, to specify the category of goals a logic of functions may be used (e.g. as in [HP91]), transformations corresponding to logical inference steps in the logic of functions.

B.2.5 The Category of Attentional Memory

Finally, the category C_{Att} of attentional knowledge is handled somewhat similarly to goals. Attentional evaluations may be modeled as maps from elements of $C_{Int} \cup C_{Dec} \cup C_{Ep} \cup C_{Proc}$ into a space V of AttentionValues. As such, attentional evaluations are functions, and may be considered as a category in a manner similar to the goal functions.

B.3 Modeling Memory Type Conversions Using Functors

Having modeled memory types as categories, we may now model conversions between memory types as mappings between categories. This is one step on the path to formalizing the notion of cognitive synergy within the formal cognitive architecture presented in the previous section.

B.3.1 Converting Between Declarative and Procedural Knowledge

To understand conversion back and forth between declarative and procedural knowledge, consider the cases:

- the category *blue* versus the procedure *isBlue* that outputs a number in $[0, 1]$ indicating the degree of blueness of its input
- the statement "the sky is blue" versus the procedure that outputs a number $[0, 1]$ indicating the degree to which its input is semantically similar to the statement "the sky is blue"
- a procedure for serving a tennis ball on the singles boundary at the edge of the service box, as close as possible to the net; versus a detailed description of this procedure, of the sort that could be communicated verbally (though it might take a long time)
- a procedure for multiplying numbers, versus a verbal description of that procedure
- a logical description of the proof of a theorem based on some axioms; versus a procedure that produces the theorem given the axioms as inputs

From these examples we can see that procedural and declarative knowledge are in a sense interchangeable; and yet, some entities seem more naturally represented procedurally, whereas other seem more naturally represented declaratively. Relatedly, it seems that some knowledge is more easily obtained via learning algorithms that operate on procedural representations; and other

knowledge is more easily obtained via learning algorithms that operate on declarative representations.

Formally, we may define a "procedure declaratization" as a functor from K_{Proc} to K_{Dec}; in other words, a pair of mappings (r, s) so that

- r maps each object in K_{Proc} into some object in K_{Dec}
- s maps each morphism $f_{Proc,i}$ in K_{Proc} into some morphism in K_{Dec}, in a way that obeys $s(f_{Proc,i} \circ f_{Proc,j}) = s(f_{Proc,i}) \circ s(f_{Proc,j})$

Similarly, we may define a "declaration procedurization" as a functor from K_{Dec} to K_{Proc}.

B.3.2 Symbol Grounding: Converting Between Episodic and Declarative Knowledge

Next we consider converting back and forth between episodic and declarative knowledge. Particular cases of this conversion have received significant attention in the cognitive science literature, referred to by the term "symbol grounding."

It is relatively straightforward to define "episode declaratization" and "declaration episodization" functors formally in the manner of the above definitions regarding declarative/procedural conversion. Conceptually,

- Episode declaratization produces a declaration describing an episode-set (naturally this declaration may be a conjunction of many simple declarations)
- Declaration episodization produces an episode-set defined as the set of episodes whose descriptions include a certain declaration

As a very simple example of declaration episodization: the predicate $isCat(x)$ could be mapped into the fuzzy set E of episodes containing cats, where the degree of membership of e in E could be measured as the degree to which e contains a cat. In this case, the episode-set would commonly be called the "grounding" of the predicate. Similarly, a relationship such as a certain sense of the preposition "with" could be mapped into the set of episodes containing relationships between physical entities that embody this word-sense.

As a very simple example of episode declaratization: an episode that an agent experienced while playing fetch with someone, could be mapped into a description of the episode including information about the kind of ball being used in the "fetch" game, the name and other properties of the other person participating in the "fetch" game, the length of time the game lasted, etc.

B.3.2.1 Algorithmically Performing Episodic/Declarative Conversion

One way that these processes could occur in an intelligent system would be for episode declaratization to guide both processes. That is, the system would need some capability to abstract declarative knowledge from observed or remembered episodes. Then, given a description, the system could carry out declaration episodization via solving the "inverse problem" of episode declaratization, i.e. given a declarative object D

1. First it could search episodic memory for episodes E_i whose (stored or on-the-fly-computed) descriptions fully or approximately match D
2. If any of the E_i is extremely accurately describable by D, then it is returned as the answer
3. Otherwise, if some of the E_i are moderately but not extremely accurately describable by D, they are used as initial guesses for local search aimed at finding some episode E whose description closely matches D
4. If no sufficiently promising E_i can be found, then a more complex cognitive process is carried out, for instance in an CogPrime system,
   - Inference may be carried out to find E_i that lead to descriptions D_i that are inferentially found to be closely equivalent to D (in spite of this near-equivalence not being obvious without inference)
   - Evolutionary learning may be carried out to "evolve" episodes, with the fitness function defined in terms of describability by D

B.3.2.2 Development of Better Symbol Groundings as Natural Transformation

As an application of the modeling of memory types as categories, it's interesting to think about the interpretation of functor categories and natural transformations in the context of memory types, and in particular in the context of "symbol groundings" of declarative knowledge in episodic knowledge.

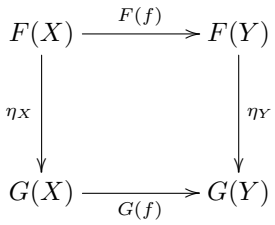
First of all, the functor category $(C_{Ep})_{C_{Dec}}$

- has as objects all functors from C_{Dec} to C_{Ep} (e.g. all methods of assigning experiences to the sets of declarations satisfying them, which nicely map transformation paths into transformation paths)
- has as morphisms the natural transformations between these functors.

That is, suppose F and G are functors between C_{Dec} and C_{Ep} ; that is, F and G are two different ways of grounding declarative knowledge in episodic knowledge. Then, a natural transformation η from F to G associates to every object X in C_{Dec} (i.e., to every declaration X) a morphism $\eta_X : F(X) \rightarrow$

$G(X)$ in C_{Ep} (that is, η_X is a composite transformation mapping the episode-set $F(X)$ into the episode-set $G(X)$) so that: for every morphism $f : X \to Y$ in C_{Dec} we have $\eta_Y \circ F(f) = G(f) \circ \eta_X$.

An easier way to conceptualize this may be to note that in the commutative diagram



we have a situation where

- X and Y represent declarations
- f represents a sequence of atomic transformations between declarations
- all corners of the diagram correspond to episode-sets
- all arrows correspond to sequences of atomic transformations between episode-sets
- η_X and η_Y represent sequences of atomic transformations between episode-sets

In other words, a natural transformation between two methods of grounding is: a mapping that assigns to each declaration, a morphism on episodic memory that preserves the commutative diagram with respect to the two methods of grounding

Cognitively, what this suggest is that developing better and better groundings is a matter of starting with one grounding and then naturally transforming it into better and better groundings.

To make things a little clearer, we now present the above commutative diagram using a more transparent, application-specific notation. Let us consider a specific example wherein:

- X is represented by the predicate $isTiger$, Y is represented by the predicate $isCat$
- f is represented by an example inference trail (i.e. transformation process) leading from $isTiger$ to $isCat$, which we will denote $isa(tiger, cat)$
- F and G are relabeled $Grounding_1$ and $Grounding_2$ (as these are two functors that ground declarative knowledge in episodic knowledge)
- $F(X)$ is relabeled $TigerEpisodes_1$ (as it's the set of episodes associated with $isTiger$ under the grounding $Grounding_1$; similarly, $F(Y)$ is relabeled $CatEpisodes_1$, $G(X)$ is relabeled $TigerEpisodes_2$, and $G(Y)$ is relabeled $CatEpisodes_2$
- $F(f)$ is relabeled $Grounding_1(isa(tiger, cat))$; and $G(f)$ is relabeled $Grounding_2(isa(tiger, cat))$

- η_X and η_Y become $\eta_{isTiger}$ and η_{isCat} respectively

With these relabelings, the above commutative diagram looks like

One may draw similar diagrams involving the other pairs of memory types, with similar interpretations.

B.3.3 Converting Between Episodic and Procedural Knowledge

Mapping between episodic and procedural knowledge may be done indirectly via the mappings already described above. Of course, such mappings could also be constructed directly but for our present purposes, the indirect approach will suffice.

Episode procedurization maps an episode-set into the set of procedures whose execution is part of the description of the episode-set. A simple example of episode procedurization would be: mapping a set of episodes involving playing "fetch" into procedures for coordinating the fetch game, throwing an object, catching an object, walking, and so forth.

Procedure episodization maps a procedure into the set of episodes appearing to contain executions of the procedure. For instance, a procedure for playing fetch would map into a set of episodes involving playing fetch; or, a procedure for adding numbers would map into a set of episodes involving addition, which might include a variety of things such as:

- "textbook examples" such as: a set of two apples, and a set of three apples, merging to form a set of five apples
- a financial transaction at a cash register in a store, involving the purchase of several items and the summing of their prices into a composite price

B.3.4 Converting Intentional or Attentional Knowledge into Declarative or Procedural Knowledge

Attentional valuations and goals are considered as functions, thus, though they may be represented in various "native" forms, their conversion into procedural knowledge is conceptually straightforward.

Conversion to declarative knowledge may occur by way of procedural knowledge, or may be more easily considered directly in some cases. For instance, the assignment of attention values to declarative knowledge items is easily represented as declarative knowledge, i.e. using statements of the form "Knowledge item K_1 has attention value V_1."

B.3.5 Converting Episodic Knowledge into Intentional or Attentional Knowledge

Episodes may contain implicit information about which entities should be attended in which contexts, and which goals have which subgoals in which contexts. Mining this information is not a simple process and requires application of significant intelligence.

B.4 Metrics on Memory Spaces

Bringing together the ideas from the previous sections, we now explain how to use the above ideas to define geometric structures for cognitive space, via defining two metrics on the space of *memory store dynamic states*. Specifically, we define the dynamic state or *d-state* of a memory store (e.g. attentional, procedural, etc.) as the series of states of that memory store (as a whole) during a time-interval. Generally speaking, it is necessary to look at d-states rather than instantaneous memory states because sometimes memory systems may store information using dynamical patterns rather than fixed structures.

It's worth noting that, according to the metrics introduced here, the above-described mappings between memory types are topologically continuous, but involve considerable geometric distortion – so that e.g., two procedures that are nearby in the procedure-based mindspace, may be distant in the declarative-based mindspace. This observation will lead us to the notion of cognitive synergy, below.

B.4.1 Information Geometry on Memory Spaces

Our first approach involves viewing memory store d-states as probability distributions. A d-state spanning time interval (p, q) may be viewed as a mapping whose input is the state of the world and the other memory stores during a given interval of time (r, s), and whose output is the state of the memory itself during interval (t, u). Various relations between these endpoints may be utilized, achieving different definitions of the mapping e.g. $p = r = t, q = s = u$ (in which case the d-state and its input and output are contemporaneous) or else $p = r, q = s = t$ (in which case the output occurs after the simultaneous d-state and input), etc. In many cases this mapping will be stochastic. If one assumes that the input is an *approximation* of the state of the world and the other memory stores, then the mapping will nearly always be stochastic. So in this way, we may model the total contents of a given memory store at a certain point in time as a probability distribution. And the process of learning is then modeled as one of *coupled changes in multiple memory stores*, in such a way as to enable ongoingly improved achievement of system goals.

Having modeled memory store states as probability distributions, the problem of measuring distance between memory store states is reduced to the problem of measuring distance between probability distributions. But this problem has a well-known solution: the Fisher-Rao metric!

Fisher information is a statistical quantity which has a a variety of applications, ranging beyond statistical data analysis, including physics [Fri98], psychology and AI [AN00]. Put simply, FI is a formal way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ upon which the probability of X depends. FI forms the basis of the Fisher-Rao metric, which has been proved the only Riemannian metric on the space of probability distributions satisfying certain natural properties regarding invariance with respect to coordinate transformations. Typically θ in the FI is considered to be a real multidimensional vector; however, [Dab99] has presented a FI variant that imposes basically no restrictions on the form of θ, which is what we need here.

Suppose we have a random variable X with a probability function $f(X, \theta)$ that depends on a parameter θ that lives in some space M that is not necessarily a dimensional space. Let $E \subseteq \mathcal{R}$ have a limit point at $t \in \mathcal{R}$, and let $\gamma : E \to M$ be a path. We may then consider a function $G(t) = \ln f(X, \gamma(t))$; and, letting $\gamma(0) = \theta$, we may then define the *generalized Fisher information* as $\mathcal{I}(\theta)_\gamma = \mathcal{I}_X(\theta)_\gamma = E\left[\left(\left(\frac{\partial}{\partial t} \ln f(X; \gamma(t))\right)^2\right) | \theta\right]$.

Next, Dabak [Dab99] has shown that the geodesic between θ and θ' is given by the exponential weighted curve $(\gamma(t))(x) = \frac{f(x,\theta)^{1-t} f(x,\theta')^t}{\int f(y,\theta)^{1-t} f(y,\theta')^t dy}$, under the weak condition that the log-likelihood ratios with respect to $f(X, \theta)$ and $f(X, \theta')$ are finite. It follows that if we use this form of curve, then the generalized Fisher information reduces properly to the Fisher information in the case of dimensional spaces. Also, along this sort of curve, the sum of

the Kullback-Leibler distances between θ and θ', known as the J-divergence, equals the integral of the Fisher information along the geodesic connecting θ and θ'.

Finally, another useful step for our purposes is to bring Fisher information together with imprecise and indefinite probabilities as discussed in [GIGH08]. For instance an indefinite probability takes the form $((L, U), k, b)$ and represents an envelope of probability distributions, whose means after k more observations lie in (L, U) with probability b. The Fisher-Rao metric between probability distributions is naturally extended to yield a metric between indefinite probability distributions.

B.4.2 Algorithmic Distance on Memory Spaces

A conceptually quite different way to measure the distance between two d-states, on the other hand, is using algorithmic information theory. Assuming a fixed Universal Turing Machine M, one may define $H(S_1, S_2)$ as the length of the shortest self-delimiting program which, given as input d-state S_1, produces as output d-state S_2. A metric is then obtained via setting $d(S_1, S_2) = (H(S_1, S_2) + H(S_2, S_1)/2$. This tells you the computational cost of transforming S_1 into S_2.

There are variations of this which may also be relevant; for instance [YGSS10] defines the generalized complexity criterion $K_\Phi(x) = min_{i \in N}\{\Phi(i, \tau_i)|L(p_i)) = x\}$, where L is a programming language, p_i is the i'th program executable by L under an enumeration in order of nonincreasing program length, τ_i is the execution time of the program p_i, $L(x)$ is the result of L executing p_i to obtain output x, and Φ is a function mapping pairs of integers into positive reals, representing the trade-off between program length and memory. Via modulating Φ, one may cause this complexity criterion to weight only program length (like standard algorithmic information theory), only runtime (like the speed prior), or to balance the two against each other in various ways.

Suppose one uses the generalized complexity criterion, but looking only at programs p_i that are given S_1 as input. Then $K_\Phi(S_2)$, relative to this list of programs, yields an asymmetric distance $H_\Phi(S_1, S_2)$, which may be symmetrized as above to yield $d_\Phi(S_1, S_2)$. This gives a more flexible measure of how hard it is to get to one of (S_1, S_2) from the other one, in terms of both memory and processing time.

One may discuss geodesics in this sort of algorithmic metric space, just as in Fisher-Rao space. A geodesic in algorithmic metric space has the property that, between any two points on the path, the *integral of the algorithmic complexity* incurred while following the path is less than or equal to that which would be incurred by following any other path between those two points. The algorithmic metric is not equivalent to the Fisher-Rao metric, a

fact that is consistent with Cencov's Theorem because the algorithmic metric is not Riemannian (i.e. it is not locally approximated by a metric defined via any inner product).

B.5 Three Hypotheses About the Geometry of Mind

Now we present three hypotheses regarding generally intelligent systems, using the conceptual and mathematical machinery we have built.

B.5.1 Hypothesis 1: Syntax-Semantics Correlation

The informational and algorithmic metrics, as defined above, are not equivalent nor necessarily closely related; however, we hypothesize that on the whole, systems will operate more intelligently if the two metrics are well correlated, implying that geodesics in one space should generally be relatively short paths (even if not geodesics) in another.

This hypothesis is a more general version of the "syntax-semantics correlation" property studied in [Loo06] in the context of automated program learning. There, it is shown empirically that program learning is more effective when programs with similar syntax also have similar behaviors. Here, we are suggesting that an intelligent system will be more effective if memory stores with similar structure and contents lead to similar effects (both externally to the agent, and on other memory systems). Hopefully the basic reason for this is clear. If syntax-semantics correlation holds, then learning based on the internal properties of the memory store, can help figure out things about the external effects of the memory store. On the other hand, if it doesn't hold, then it becomes quite difficult to figure out how to adjust the internals of the memory to achieve desired effects.

The assumption of syntax-semantics correlation has huge implications for the design of learning algorithms associated with memory stores. All of CogPrime 's learning algorithms are built on this assumption. For, example CogPrime 's MOSES procedure learning component [Loo06] assumes syntax-semantics correlation for individual programs, from which it follows that the property holds also on the level of the whole declarative memory store. And CogPrime 's PLN probabilistic inference component [GIGH08] uses an inference control mechanism that seeks to guide a new inference via analogy to prior similar inferences, thus embodying an assumption that structurally similar inferences will lead to similar behaviors (conclusions).

B.5.2 Hypothesis 2: Cognitive Geometrodynamics

In general relativity theory there is the notion of "geometrodynamics," referring to the feedback by which matter curves space, and then space determines the movement of matter (via the rule that matter moves along geodesics in curved spacetime) [MTW73]. One may wonder whether an analogous feedback exists in cognitive geometry. We hypothesize that the answer is yes, to a limited extent. On the one hand, according to the above formalism, the curvature of mindspace is induced by the knowledge in the mind. On the other hand, one may view cognitive activity as approximately following geodesics in mindspace.

Let's say an intelligent system has the goal of producing knowledge meeting certain characteristics (and note that the desired achievement of a practical system objective may be framed in this way, as seeking the true knowledge that the objective has been achieved). The goal then corresponds to some set of d-states for some of the mind's memory stores. A simplified but meaningful view of cognitive dynamics is, then, that the system seeks the shortest path from the current d-state to the region in d-state space comprising goal d-states. For instance, considering the algorithmic metric, this reduces to the statement that at each time point, the system seeks to move itself along a path toward its goal, in a manner that requires the minimum computational cost – i.e. along some algorithmic geodesic. And if there is syntax-semantics correlation, then this movement is also approximately along a Fisher-Rao geodesic.

And as the system progresses from its current state toward its goal-state, it is creating new memories – which then curve mindspace, possibly changing it substantially from the shape it had before the system started moving toward its goal. This is a feedback conceptually analogous to, though in detail very different from, general-relativistic geometrodynamics.

There is some subtlety here related to fuzziness. A system's goals may be achievable to various degrees, so that the goal region may be better modeled as a fuzzy set of lists of regions. Also, the system's current state may be better viewed as a fuzzy set than as a crisp set. This is the case with CogPrime , where uncertain knowledge is labeled with confidence values along with probabilities; in this case the confidence of a logical statement may be viewed as the fuzzy degree with which it belongs to the system's current state. But this doesn't change the overall cognitive-geometrodynamic picture, it just adds a new criterion; one may say that the cognition seeks a geodesic from a high-degree portion of the current-state region to a high-degree portion of the goal region.

B.5.3 Hypothesis 3: Cognitive Synergy

Cognitive synergy, discussed extensively above, is a conceptual explanation of what makes it possible for certain sorts of integrative, multi-component cognitive systems to achieve powerful general intelligence [Goe09a]. The notion pertains to systems that possess knowledge creation (i.e. pattern recognition / formation / learning) mechanisms corresponding to each multiple memory types. For such a system to display cognitive synergy, each of these cognitive processes must have the capability to recognize when it lacks the information to perform effectively on its own; and in this case, to dynamically and interactively draw information from knowledge creation mechanisms dealing with other types of knowledge. Further, this cross-mechanism interaction must have the result of enabling the knowledge creation mechanisms to perform much more effectively in combination than they would if operated non-interactively.

How does cognitive synergy manifest itself in the geometric perspective we've sketched here? Perhaps the most straightforward way to explore it is to construct a composite metric, merging together the individual metrics associated with specific memory spaces.

In general, given N metrics $d_k(x,z), k = 1 \ldots N$ defined on the same finite space M, we can define the "min-combination" metric

$$d_{d_1,\ldots,d_N}(x,z) = min_{y_0=x,y_{n+1}=z,y_i \in M, r(i) \in \{1,\ldots,N\}, i \in \{1,\ldots,n\}, n \in \mathbb{Z}} \sum_{i=0}^{n} d_{r(i)}(y_i, y_{i+1})$$

This metric is conceptually similar to (and mathematically generalizes) min-cost metrics like the Levenshtein distance used to compare strings [Lev66]. To see that it obeys the metric axioms is straightforward; the triangle inequality follows similarly to the case of the Levenshtein metric. In the case where M is infinite, one replaces min with inf (the infimum) and things proceed similarly. The min-combination distance from x to z tells you the length of the shortest path from x to z, using the understanding that for each portion of the path, one can choose any one of the metrics being combined. Here we are concerned with cases such as $d_{syn} = d_{d_{Proc}, d_{Dec}, d_{Ep}, d_{Att}}$.

We can now articulate a geometric version of the principle of cognitive synergy. Basically: cognitive synergy occurs when the synergetic metric yields significantly shorter distances between relevant states and goals than any of the memory-type-specific metrics. Formally, one may say that:

Definition B.1. An intelligent agent A (modeled by SRAM) displays **cognitive synergy** to the extent

$$syn(A) \equiv$$

$$\int \left(d_{synergetic}(x,z) - min\left(d_{Proc}(x,z), d_{Dec}(x,z), d_{Ep}(x,z), d_{Att}(x,z)\right)\right) d\mu(x) d\mu(z)$$

where μ measures the relevance of a state to the system's goal-achieving activity.

B.6 Next Steps in Refining These Ideas

These ideas may be developed in both practical and theoretical directions. On the practical side, we have already had an interesting preliminary success, described briefly in 23 where we show that (in some small examples at any rate) replacing CogPrime 's traditional algorithm for attentional learning with an explicitly information-geometric algorithm leads to dramatic increases in the intelligence of the attentional component. This work needs to be validated via implementation of a scalable version of the information geometry algorithm in question, and empirical work also needs to be done to validate the (qualitatively fairly clear) syntax-semantics correlation in this case. But tentatively, this seems to be an early example of improvement to an AGI system resulting from modifying its design to more explicitly exploit the mind-geometric principles outlined here.

Potentially, each of the inter-cognitive-process synergies implicit in the CogPrime design may be formalized in the geometric terms outlined here, and doing so is part of our research programme going forward.

More generally, on the theoretical side, a mass of open questions looms. The geometry of spaces defined by the min-combination metric is not yet well-understood, and neither is the Fisher-Rao metric over nondimensional spaces or the algorithmic metric (especially in the case of generalized complexity criteria). Also the interpretation of various classes of learning algorithms in terms of cognitive geometrodynamics is a subtle matter, and may prove especially fruitful for algorithms already defined in probabilistic or information-theoretic terms.

B.7 Returning to Our Basic Claims about CogPrime

Finally, we return to the list of basic claims about CogPrime given at the end of Chapter 1, and review their connection with the ideas in this appendix. Not all of the claims there are directly related to the ideas given here, but many of them are; to wit:

6. It is most effective to teach an AGI system aimed at roughly human-like general intelligence via a mix of spontaneous learning and explicit instruction, and to instruct it via a combination of imitation, reinforce-

ment and correction, and a combination of linguistic and nonlinguistic instruction

- **Mindspace interpretation**. Different sorts of learning are primarily focused on different types of memory, and hence on different mindspaces. The effectiveness of learning focused on a particular memory type depends on multiple factors including: the general competence of the agent's learning process corresponding to that memory store, the amount of knowledge already built up in that memory store, and the degree of syntax-semantics correlation corresponding to that memory store. In terms of geometrodynamics, learning in a manner focused on a certain memory type, has significant impact in terms of reshaping the mindspace implied by that memory store.

7. One effective approach to teaching an AGI system human language is to supply it with some in-built linguistic facility, in the form of rule-based and statistical-linguistics-based NLP systems, and then allow it to improve and revise this facility based on experience
   - **Mindspace interpretation**. Language learning purely in declarative space (formal grammar rules), or purely in attentional space (statistical correlations between linguistic inputs), or purely in episodic or procedural space (experiential learning), will not be nearly so effective as language learning which spans multiple memory spaces. Language learning (like many other kinds of humanly natural learning) is better modeled as cognitive-synergetic cognitive geometrodynamics, rather than as single-memory-type cognitive geometrodynamics.
8. An AGI system with adequate mechanisms for handling the key types of knowledge mentioned above, and the capability to explicitly recognize large-scale pattern in itself, should, **upon sustained interaction with an appropriate environment in pursuit of appropriate goals**, emerge a variety of complex structures in its internal knowledge network, including (but not limited to)
   - a hierarchical network, representing both a spatiotemporal hierarchy and an approximate "default inheritance" hierarchy, cross-linked
   - a heterarchical network of associativity, roughly aligned with the hierarchical network
   - a self network which is an approximate micro image of the whole network
   - inter-reflecting networks modeling self and others, reflecting a "mirrorhouse" design pattern

   What does this mean geometrically?
   - **Mindspace interpretation**. The self network and mirrorhouse networks imply a roughly fractal structure for mindspace, especially

when considered across multiple memory types (since the self network spans multiple memory types). Peripherally, it's interesting that the physical universe has a very roughly fractal structure too, e.g. with solar systems within galaxies within galactic clusters; so doing geometrodynamics in roughly fractal curved spaces is not a new idea.

9. Given the strengths and weaknesses of current and near-future digital computers,
    a. A (loosely) neural-symbolic network is a good representation for directly storing many kinds of memory, and interfacing between those that it doesn't store directly
        - **Mindspace interpretation**. The "neural" aspect stores associative knowledge, and the "symbolic" aspect stores declarative knowledge; and the superposition of the two in a single network makes it convenient to implement cognitive processes embodying cognitive synergy between the two types of knowledge.
    b. Uncertain logic is a good way to handle declarative knowledge
        - **Mindspace interpretation**. There are many senses in which uncertain logic is "good" for AGI; but the core points are that:
            - it makes representation of real-world relationships relatively compact
            - it makes inference chains of real-world utility relatively short
            - it gives high syntax-semantics correlation for logical relationships involving uncertainty (because it lends itself to syntactic distance measures that treat uncertainty naturally, gauging distance between two logical relationships based partly on the distances between the corresponding uncertainty values; e.g. the PLN metric defined in terms of SimilarityLink truth values)
            - because the statistical formulas for truth value calculation are related to statistical formulas for association-finding, it makes synergy between declarative and associative knowledge relatively straightforward.
    c. Programs are a good way to represent procedures (both cognitive and physical-action, but perhaps not including low-level motor-control procedures)
    d. Evolutionary program learning is a good way to handle difficult program learning problems
        - Probabilistic learning on normalized programs is one effective approach to evolutionary program learning
        - MOSES is one good realization
            - **Mindspace interpretation**. Program normalization creates relatively high syntax-semantics correlation in procedural knowledge (program) space, and MOSES is an algorithm that systematically exploits this knowledge.

e. Multistart hill-climbing on normalized programs, with a strong Occam prior, is a good way to handle relatively straightforward program learning problems
f. Activation spreading is a reasonable way to handle attentional knowledge (though other approaches, with greater overhead cost, may provide better accuracy and may be appropriate in some situations)
   - Artificial economics is an effective approach to activation spreading in the context of neural-symbolic network.
   - ECAN is one good realization, with Hebbian learning as one route of learning associative relationships, and more sophisticated methods such as information-geometric ones potentially also playing a role
   - A good trade-off between comprehensiveness and efficiency is to focus on two kinds of attention: processor attention (represented in CogPrime by ShortTermImportance) and memory attention (represented in CogPrime by LongTermImportance)

   The **mindspace interpretation** includes the observations that
   - Artificial economics provides more convenient conversion between attentional and declarative knowledge, compared to more biologically realistic neural net type models of attentional knowledge
   - In one approach to structuring the attentional mindspace, historical knowledge regarding what was worth attending (i.e. high-strength HebbianLinks between Atoms that were in the AttentionalFocus at the same time, and linkages between these maps and system goals) serves to shape the mindspace, and learning the other HebbianLinks in the network may be viewed as an attempt to follow short paths through attentional mindspace (as explicitly shown in Chapter 23).
g. Simulation is a good way to handle episodic knowledge (remembered and imagined)
   - Running an internal "world simulation engine" is an effective way to handle simulation

   What's the **mindspace interpretation**? For example,
   - The world simulation engine takes a certain set of cues and scattered memories related to an episode, and creatively fills in the gaps to create a full-fledged simulation of the episode. Syntax-semantics correlation means that stating "sets of cues and scattered memories" A and B are similar, is approximately the same as stating that the corresponding full-fledged simulations are similar.
   - Many dreams seem to be examples of following paths through episode space, from one episode to another semantically related one, etc. But these paths are often aimless, though generally following semantic similarity. Trying to think of or remember an

episode matching certain constraints, is a process where following short paths through episodic mindspace is relevant.

h. Hybridization of one's integrative neural-symbolic system with a spatiotemporally hierarchical deep learning system is an effective way to handle representation and learning of low-level sensorimotor knowledge
   - DeSTIN is one example of a deep learning system of this nature that can be effective in this context

   The **mindspace interpretation** includes the observations that
   - Linkages between the internal nodes of DeSTIN and the
   - Spatio-temporally hierarchical perception-action systems like DeSTIN have high syntax-sematics correlation for sensory knowledge, as they embody the spatiotemporally hierarchical structure of the perceived world

i. One effective way to handle goals is to represent them declaratively, and allocate attention among them economically
   - CogPrime 's PLN/ECAN based framework for handling intentional knowledge is one good realization

   One aspect of the **mindspace interpretation** is that using PLN and ECAN together to represent goals, aids with the cognitive synergy between declarative, associative and intentional space. Achieving a goal is then (among other things) about finding short paths to the goal thru declarations, associations and actions.

10. It is important for an intelligent system to have some way of recognizing large-scale patterns in itself, and then embodying these patterns as new, localized knowledge items in its memory

    - Given the use of a neural-symbolic network for knowledge representation, a graph-mining based "map formation" heuristic is one good way to do this

    Key aspects of the **mindspace interpretation** are that:

    - via map formation, associative (map) and declarative, procedural or episodic (localized) knowledge are correlated, promoting cognitive synergy
    - approximate and emergent inference on concept maps, occurring via associational processes, roughly mirrors portions of PLN reasoning on declarative concepts and relationships. This aids greatly with cognitive synergy, and in fact one can draw "natural transformations" (in the language of category theory) between map inference and localized, declarative concept inference.

11. Occam's Razor: Intelligence is closely tied to the creation of procedures that achieve goals in environments *in the simplest possible way.*

- Each of an AGI system's cognitive algorithms should embody a "simplicity bias" in some explicit or implicit form

Obviously, one aspect of the **mindspace interpretation** of this principle is simply the geometrodynamic idea of following the shortest path through mindspace, toward the appointed set of goal states. Also, this principle is built into the definition of semantic space used in the mindspace framework developed above, since computational simplicity is used to define the semantic metric between memory items.

While the abstract "mind geometry" theory presented in this appendix doesn't (yet) provide a way of deriving the CogPrime design from first principles, it does provide a useful general vocabulary for discussing the various memory types and cognitive processes in CogPrime in a unified way. And it also has some power to suggest novel algorithms to operated within cognitive processes, as in the case of our work on information geometry and ECAN. Whether mind geometry will prove a really useful ingredient in CogPrime theory or AGI theory more broadly, remains to be determined; but we are cautiously optimistic and intend to pursue further in this direction.

Appendix C
Emergent Reflexive Mental Structures

Co-authored with Tony Smith, Onar Aam and Kent Palmer

C.1 Introduction

This appendix deals with some complex emergent structures we suspect may emerge in advanced CogPrime and other AGI systems. The ideas presented here are flatly conjectural, and we stress that the CogPrime design is not dependent thereupon. The more engineering-oriented reader may skip them without any near-term loss. However, we do believe that this sort of rigorous lateral thinking is an important part of any enterprise as ambitious as building a human-level AGI.

We have stated that the crux of an AGI system really lies on the emergent level – on the structures and dynamics that arise in the system as a result of its own self-organization and its coupling with other minds and the external world. We have talked a bit about some of these emergent patterns – e.g. maps and various sorts of networks – but by and large they have stayed in the background. In this appendix we will indulge in a bit of speculative thinking about some of the high-level emergent patterns that we believe may emerge in AGI systems once they begin to move toward human-level intelligence, and specifically once they acquire a reasonably sophisticated ability to model themselves and other minds. * These patterns go beyond the relatively well-accepted network structures reviewed in chapter **??**, and constitute an

* A note from Ben Goertzel. In connection with the material in this appendix, I would like to warmly acknowledge Louis Kauffman for an act of kindness that occurred back in 1986, when I was a 19 year old PhD student, when he mailed me a copy of his manuscript **Sign and Space**, which contained so many wonderful ideas and drawings related to the themes considered here. Lou's manuscript wasn't my first introduction to the meme of consciousness and self-reference – I got into these ideas first via reading Douglas Hofstadter at age 13 in 1979, and then later via reading G. Spencer-Brown. But my brief written correspondence with Lou (this was before email was common even in universities) and his lovely hand-written and -drawn manuscript solidified my passion for these sorts of ideas, and increased my confidence that they are not only fascinating but deeply meaningful.

edgier, more ambitious hypothesis regarding the emergent network structures of general intelligence.

More specifically, the thesis of this appendix is that there are certain abstract algebraic structures that typify the self-structure of human beings and any other intelligent systems relying on empathy for social intelligence. These structures may be modeled using various sorts of mathematics, including hypersets and also algebraic structures called quaternions and octonions (which also play a critical role in modern theoretical physics [DG94]). And, assuming mature, reasonably intelligent AGI systems are created, it will be possible to empirically determine whether the mathematical structures posited here do or do not emerge in them.

C.2 Hypersets and Patterns

The first set of hypotheses we will pursue in this appendix is that the abstract structures corresponding to free will, reflective consciousness and phenomenal self are effectively modeled using the mathematics of *hypersets*.

What are these things called hypersets, which we posit as cognitive models?

In the standard axiomatizations of set theory, such as Zermelo-Frankel set theory [**?**], there is an axiom called the Axiom of Foundation, which implies that no set can contain itself as a member. That is, it implies that all sets are "well founded" – they are built up from other sets, which in turn are built up from other sets, etc., ultimately being built up from the empty set or from atomic elements. The hierarchy via which sets are built from other sets may be infinite (according to the usual Axiom of Infinity), but it goes in only one direction – if set A is built from set B (or from some other set built from set B), then set B can't be built from set A (or from any other set built from set A).

However, since very shortly after the Axiom of Foundation was formulated, there have been various alternative axiomatizations which allow "non-well-founded" sets (aka hypersets), i.e. sets that *can* contain themselves as members, or have more complex circular membership structures. Hyperset theory is generally formulated as an extension of classical set theory rather than a replacement – i.e., the well-founded sets within a hyperset domain conform to classical set theory. In recent decades the theory of non-well-founded sets has been applied in computer science (e.g. process algebra [**?**]), linguistics and natural language semantics (situation theory [Bar89]), philosophy (work on the Liar Paradox [BE89]), and other areas.

For instance, in hyperset theory you can have

$$A = \{A\}$$

$$A = \{B, \{A\}\}$$

and so forth. Using hypersets you can have functions that take themselves as arguments, and many other interesting phenomena that aren't permitted by the standard axioms of set theory. The main work of this paper is to suggest specific models of free will, reflective consciousness and phenomenal self in terms of hyperset mathematics.

The reason the Axiom of Foundation was originally introduced was to avoid paradoxes like the Russell Set (the set of all sets that contain themselves). None of these variant set theories allow all possible circular membership structures; but they allow restricted sets of such, sculpted to avoid problems like the Russell Paradox.

One currently popular form of hyperset theory is obtained by replacing the Axiom of Foundation with the Anti-Foundation Axiom (AFA) which, roughly speaking, permits circular membership structures that map onto graphs in a certain way. All the hypersets discussed here are easily observed to be allowable under the AFA (according to the the Solution Lemma stated in [Acz88]).

Specifically, the AFA uses the notion of an *accessible pointed graph* – a directed graph with a distinguished element (the "root") such that for any node in the graph there is at least one path in the directed graph from the root to that node. The AFA states that every accessible pointed graph corresponds to a unique set. For example, the graph consisting of a single vertex with a loop corresponds to a set which contains only itself as element,

While the specific ideas presented here are novel, the idea of analyzing consciousness and related structures in terms of infinite recursions and non-foundational structures has occurred before, for instance in the works of Douglas Hofstadter [Hof79], G. Spencer-Brown [SB67], Louis Kauffmann [Kau] and Francisco Varela [Var79]. None of these works uses hypersets in particular; but a more important difference is that none of them attempts to deal with particular psychological phenomena in terms of correlation, causation, pattern theory or similar concepts; they essentially stop at the point of noting the presence of a formalizable pattern of infinite recursion in reflective consciousness. [Var79] does venture into practical psychology via porting some of R.D. Laing's psychosocial "knots" [Lai72] into a formal non-foundational language; but this is a very specialized exercise that doesn't involve modeling general psychological structures or processes. Situation semantics [Bar89] does analyze various commonsense concepts and relationships using hypersets; however, it doesn't address issues of subjective experience explicitly, and doesn't present formal treatments of the phenomena considered here.

C.2.1 Hypersets as Patterns in Physical or Computational Systems

Hypersets are large infinite sets – they are certainly not computable – and so one might wonder if a hyperset model of consciousness supports Penrose [Pen96] and Hameroff's [Ham87] notion of consciousness as involving as-yet unknown physical dynamics involving uncomputable mathematics. However, this is not our perspective.

In the following we will present a number of particular hypersets and discuss their presence as patterns in intelligent systems. But this does not imply that we are positing intelligent systems to fundamentally *be* hypersets, in the sense that, for instance, classical physics posits intelligent systems to be matter in $3+1$ dimensional space. Rather, we are positing that it is possible for hypersets to serve as *patterns* in physical systems, where the latter may be described in terms of classical or modern physics, or in terms of computation.

How is this possible? If a hyperset can *produce* a somewhat accurate model of a physical system, and is judged *simpler* than a detailed description of the physical system, then it may be a pattern in that system according to the definition of pattern given above.

Recall the definition of pattern given in chapter 3:

Definition 10 *Given a metric space (M, d), and two functions $c : M \to [0, \infty]$ (the "simplicity measure") and $F : M \to M$ (the "production relationship"), we say that $\mathcal{P} \in M$ is a* **pattern** *in $X \in M$ to the degree*

$$\iota_X^{\mathcal{P}} = \left(\left(1 - \frac{d(F(\mathcal{P}), X)}{c(X)} \right) \frac{c(X) - c(\mathcal{P})}{c(X)} \right)^+$$

This degree is called the **pattern intensity** *of $\mathcal{P}$ in X.*

To use this definition to bridge the gap between hypersets and ordinary computer programs and physical systems, we may define the metric space M to contain both hypersets and computer programs, and also tuples whose elements may be freely drawn from either of these classes. Define the partial order $<$ so that if X is an entry in a tuple T, then $X < T$.

Distance between two programs may be defined using the algorithmic information metric

$$d_I(A, B) = I(A|B) + I(B|A)$$

where $I(A|B)$ is the length of the shortest self-delimiting program for computing A given B [Cha08]. Distance between two hypersets X and Y may be defined as

$$d_H(X, Y) = d_I(g(A), g(B))$$

where $g(A)$ is the graph (A's apg, in AFA lingo) picturing A's membership relationship. If A is a program and X is a hyperset, we may set $d(A, X) = \infty$.

Next, the production relation F may be defined to act on a (hyperset,program) pair $P = (X, A)$ via feeding the graph representing X (in some standard encoding) to A as an input. According to this production relation, P may be a pattern in the bit string $B = A(g(X))$; and since $X < P$, the hyperset X may be a subpattern in the bit string B.

It follows from the above that a hyperset can be part of the mind of a finite system described by a bit string, a computer program, or some other finite representation. But what sense does this make conceptually? Suppose that a finite system S contains entities of the form

$$\begin{aligned} &C \\ &G(C) \\ &G(G(C)) \\ &G(G(G(C))) \\ &... \end{aligned}$$

Then it may be effective to compute S using a (hyperset, program) pair containing the hyperset

$$X = G(X)$$

and a program that calculates the first k iterates of the hyperset. If so, then the hyperset $\{X = G(X)\}$ may be a subpattern in S. We will see some concrete examples of this in the following.

Whether one thing is a pattern in another depends not only on production but also on relative simplicity. So, if a system is studied by an observer who is able to judge some hypersets as simpler than some computational entities, then there is the possibility for hypersets to be subpatterns in computational entities, according to that observer. For such an observer, there is the possibility to model mental phenomena like will, self and reflective consciousness as hypersets, consistently with the conceptualization of mind as pattern.

C.3 A Hyperset Model of Reflective Consciousness

Now we proceed to use hypersets to model the aspect of mind we call "reflective consciousness."

Whatever your view of the ultimate nature of consciousness, you probably agree that different entities in the universe manifest different *kinds* of consciousness or “awareness." Worms are aware in a different way than rocks;

and dogs, pigs, pigeons and people are aware in a different way from worms. In [Goe94] it is argued that hypersets can be used to model the sense in which the latter beasts are conscious whereas worms are not – i.e. what might be called "reflective consciousness."

We begin with the old cliche' that

Consciousness is consciousness of consciousness

Note that this is nicely approximated by the series

A
Consciousness of A
Consciousness of consciousness of A
...

This is conceptually elegant, but doesn't really serve as a definition or precise characterization of consciousness. Even if one replaces it with

Reflective consciousness is reflective consciousness of reflective consciousness

it still isn't really adequate as a model of most reflectively conscious experience – although it does seem to capture *something* meaningful.

In hyperset theory, one can write an equation

$$f = f(f)$$

with complete mathematical consistency. You feed f as input: f ... and you receive as output: f. But while this sort of anti-foundational recursion may be closely associated with consciousness, this simple equation itself doesn't tell you much about consciousness. We don't really want to say

ReflectiveConsciousness = ReflectiveConsciousness(ReflectiveConsciousness)

It's more useful to say:

Reflective consciousness is a hyperset, and reflective consciousness is contained in its membership scope

Here by the "membership scope" of a hyperset S, what we mean is the members of S, plus the members of the members of S, etc. However, this is no longer a definition of reflective consciousness, merely a characterization. What it says is that reflective consciousness must be defined anti-foundationally as some sort of construct via which reflective consciousness builds reflective consciousness from reflective consciousness – but it doesn't specify exactly how.

Putting this notion together with the discussion from Chapter 3 on patterns, correlations and experience, we arrive at the following working definition of reflective consciousness. Assume the existence of some formal language with enough power to represent nested logical predicates, e.g. standard predicate calculus will suffice; let us refer to expressions in this language as "declarative content." Then we may say

Definition C.1. "S is reflectively conscious of X" is defined as:
The declarative content that {"S is reflectively conscious of X" correlates with "X is a pattern in S"}

For example: Being reflectively conscious of a tree means having in one's mind declarative knowledge of the form that one's reflective consciousness of that tree is correlated with that tree being a pattern in one's overall mind-state. Figure C.1 graphically depicts the above definition.

Note that this declarative knowledge doesn't have to be *explicitly* represented in the experiencer's mind as a well-formalized language – just as pigeons, for instance, can carry out deductive reasoning without having a formalization of the rules of Boolean or probabilistic logic in their brains. All that is required is that the conscious mind has an internal "informal, possibly implicit" language capable of expressing and manipulating simple hypersets. Boolean logic is still a subpattern in the pigeon's brain even though the pigeon never explicitly applies a Boolean logic rule, and similarly the hypersets of reflective consciousness may be subpatterns in the pigeon's brain in spite of its inability to explicitly represent the underlying mathematics.

Turning next to the question of how these hyperset constructs may emerge from finite systems, Figures C.2, C.3 and C.4 show the first few iterates of a series of structures that would naturally be computed by a pattern containing as a subpattern Ben's reflective consciousness of his inner image of a money tree. The presence of a number of iterates in this sort of series, as patterns or subpatterns in Ben, will lead to the presence of the hyperset of "Ben's reflective consciousness of his inner image of a money tree" as a subpattern in his mind.

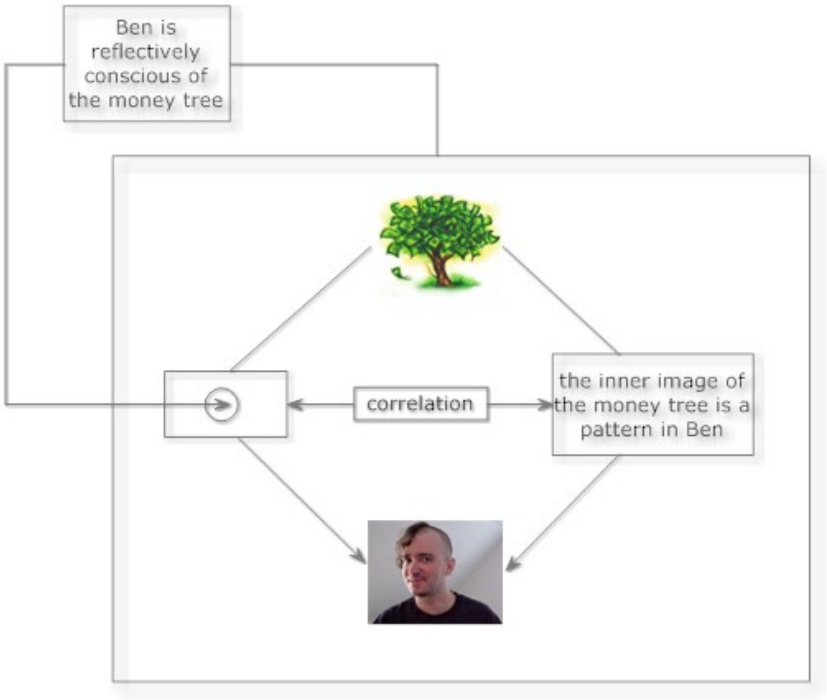


Fig. C.1 Graphical depiction of "Ben is reflectively conscious of his inner image of a money tree"

C.4 A Hyperset Model of Will

The same approach can be used to define the notion of "will," by which is meant the sort of willing process that we carry out in our minds when we subjectively feel like we are deciding to make one choice rather than another [Wal01].

In brief:

Definition C.2. "S wills X" is defined as:
The declarative content that {"S wills X" causally implies "S does X"}

Figure C.5 graphically depicts the above definition.

To fully explicate this is slightly more complicated than in the case of reflective consciousness, due to the need to unravel what's meant by "causal implication." For sake of the present discussion we will adopt the view of causation presented in [GMIH08], according to which *causal implication* may be defined as: Predictive implication combined with the existence of a plausible causal mechanism.

More precisely, if A and B are two classes of events, then A "predictively implies B" if it's probabilistically true that in a situation where A occurs, B

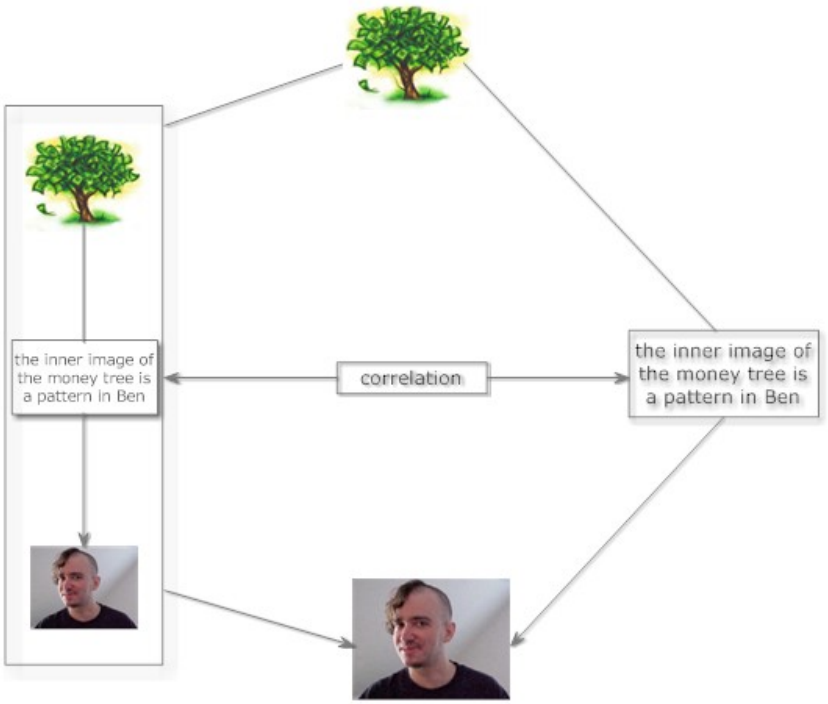


Fig. C.2 First iterate of a series that converges to Ben's reflective consciousness of his inner image of a money tree

often occurs afterwards. (Of course, this is dependent on a model of what is a "situation", which is assumed to be part of the mind assessing the predictive implication.)

And, a "plausible causal mechanism" associated with the assertion "A predictively implies B" means that, if one removed from one's knowledge base all specific instances of situations providing direct evidence for "A predictively implies B", then the inferred evidence for "A predictively implies B" would still be reasonably strong. (In PLN lingo, this means there is strong intensional evidence for the predictive implication, along with extensional evidence.)

If X and Y are particular events, then the probability of "X causally implies Y" may be assessed by probabilistic inference based on the classes (A, B, etc.) of events that X and Y belong to.

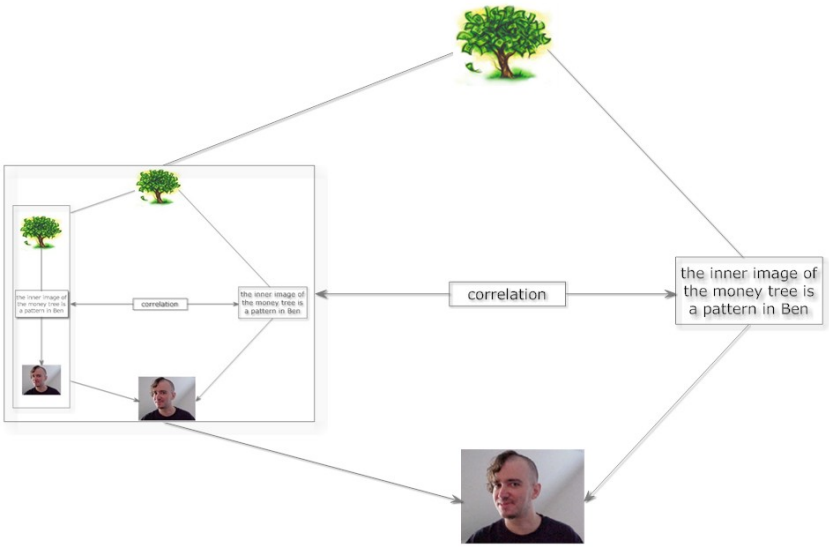


Fig. C.3 Second iterate of a series that converges to Ben's reflective consciousness of his inner image of a money tree

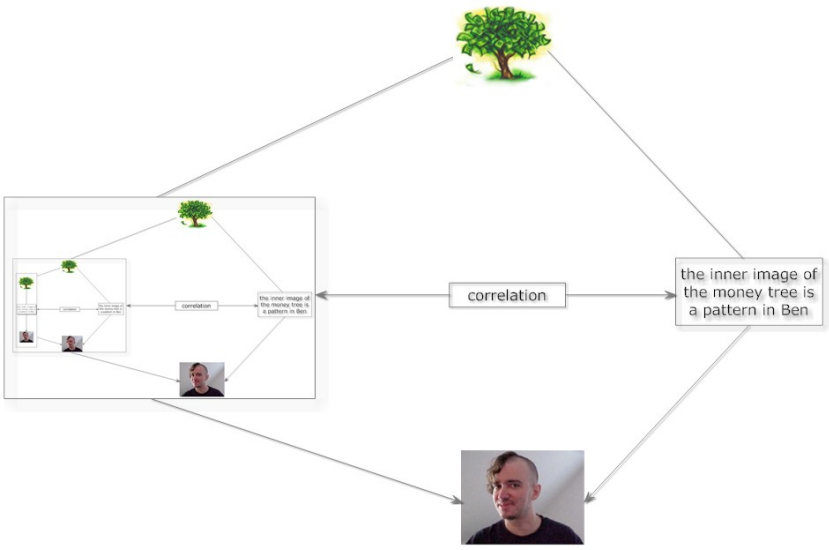


Fig. C.4 Third iterate of a series that converges to Ben's reflective consciousness of his inner image of a money tree

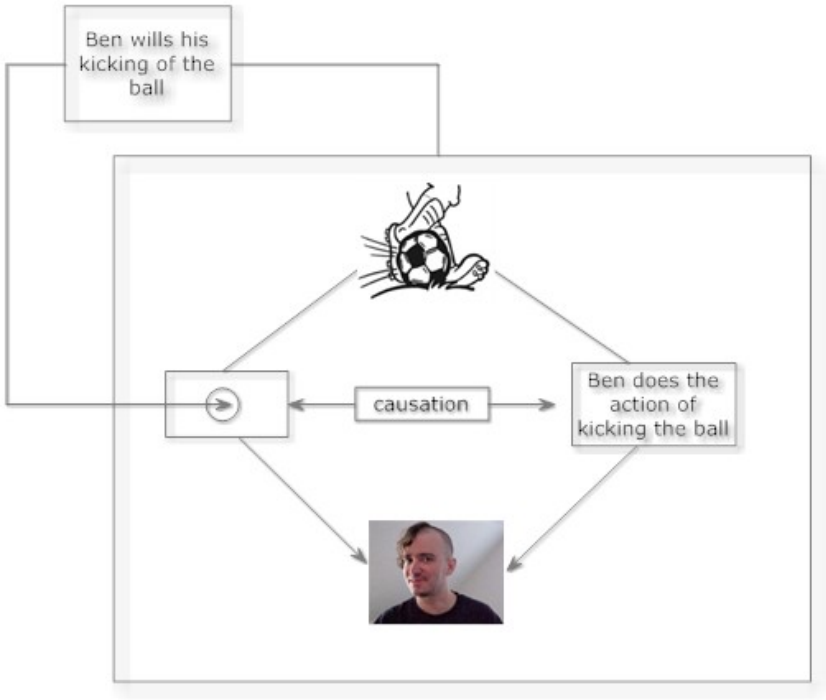


Fig. C.5 Graphical depiction of "Ben wills himself to kick the soccer ball"

C.4.1 In What Sense Is Will Free?

Briefly, what does this say about the philosophical issues traditionally associated with the notion of "free will"?

It doesn't suggest any validity for the idea that will somehow adds a magical ingredient beyond the familiar ingredients of "rules" plus "randomness." In that sense, it's not a very radical approach. It fits in with the modern understanding that free will is to a certain extent an "illusion", and that some sort of "natural autonomy" [Wal01] is a more realistic notion.

However, it also suggests that "illusion" is not quite the right word. An act of will may have causal implication, according to the psychological definition of the latter, without this action of will violating the notion of deterministic/stochastic equations of the universe. The key point is that causality is itself a psychological notion (where within "psychological" I include cultural as well as individual psychology). Causality is not a physical notion; there is no branch of science that contains the notion of causation within its formal language. In the internal language of mind, acts of will have causal impacts – and this is consistent with the hypothesis that mental actions may potentially be ultimately determined via deterministic/stochastic lower-level dynamics. Acts

of will exist on a different level of description than these lower-level dynamics. The lower-level dynamics are part of a theory that compactly explains the behavior of cells, molecules and particles; and some aspects of complex higher-level systems like brains, bodies and societies. Will is part of a theory that compactly explains the decisions of a mind to itself.

C.4.2 Connecting Will and Consciousness

Connecting back to reflective consciousness, we may say that:

> In the domain of reflective conscious experiences, acts of will are experienced as causal.

This may seem a perfectly obvious assertion. What's nice is that, in the present perspective, it seems to fall out of a precise, abstract characterization of consciousness and will.

C.5 A Hyperset Model of Self

Finally, we posit a similar characterization for the cognitive structure called the "phenomenal self" – i.e. the psychosocial model that an organism builds of itself, to guide its interaction with the world and also its own internal choices. For a masterfully thorough treatment of this entity, see Thomas Metzinger's book *Being No One* [Met04]).

One way to conceptualize self is in terms of the various forms of memory comprising a humanlike intelligence [TC05], which include procedural, semantic and episodic memory.

In terms of procedural memory, an organism's phenomenal self may be viewed as a *predictive model* of the system's behavior. It need not be a wholly accurate predictive model; indeed many human selves are wildly inaccurate, and aesthetically speaking, this can be part of their charm. But it is a predictive model that the system uses to predict its behavior.

In terms of declarative memory, a phenomenal self is used for explanation – it is an *explanatory model* of the organism's behaviors. It allows the organism to carry out (more or less uncertain) inferences about what it has done and is likely to do.

In terms of episodic memory, a phenomenal self is used as the protagonist of the organism's remembered and constructed narratives. It's a fictional character, "based on a true story," simplified and sculpted to allow the or-

ganism to tell itself and others (more or less) sensible stories about what it does.

The simplest version of a hyperset model of self would be:

Definition C.3. "X is part of S's phenomenal self" is defined as the declarative content that
{"X is part of S's phenomenal self" correlates with "X is a persistent pattern in S over time"}

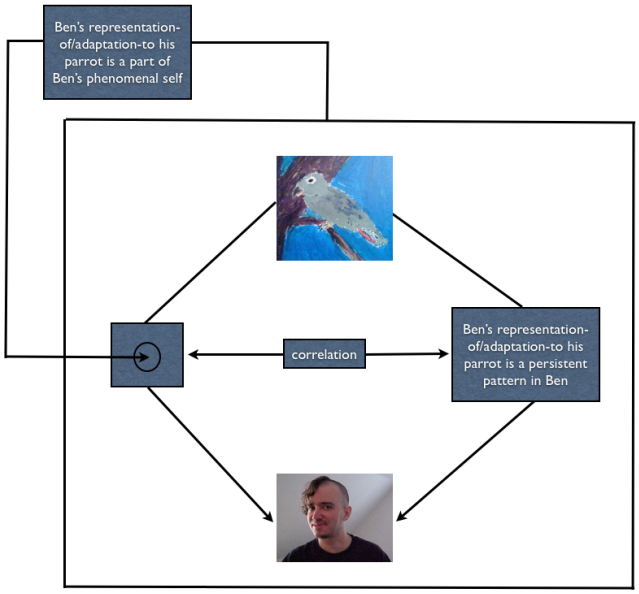


Fig. C.6 Graphical depiction of "Ben's representation-of/adaptation to his parrot is a part of his phenomenal self" (*Image of parrot is from a painting by Scheherazade Goertzel*)

Figure C.6 graphically depicts the above definition.

A subtler version of the definition would take into account the multiplicity of memory types:

Definition C.4. "X is part of S's phenomenal self" is defined as the declarative content that
{"X is a part of S's phenomenal self" correlates with "X is a persistent pattern in S's declarative, procedural and episodic memory over time"}

One thing that's nice about this definition (in both versions) is the relationship that it applies between self and reflective consciousness. In a formula:

Self is to long-term memory as reflective consciousness is to short-term memory

According to these definitions:

- A mind's self is nothing more or less than its reflective consciousness of its persistent being.
- A mind's reflective consciousness is nothing more or less than the self of its short-term being.

C.6 Validating Hyperset Models of Experience

We have made some rather bold hypotheses here, regarding the abstract structures present in physical systems corresponding to the experiences of reflective consciousness, free will and phenomenal self. How might these hypotheses be validated or refuted?

The key is the evaluation of hypersets as subpatterns in physical systems. Taking reflective consciousness as an example, one could potentially validate whether, when a person is (or, in the materialist view, reports being) reflectively conscious of a certain apple being in front of them, the hypothetically corresponding hyperset structure is actually a subpattern in their brain structure and dynamics. We cannot carry out this kind of data analysis on brains yet, but it seems within the scope of physical science to do so.

But, suppose the hypotheses presented here are validated, in the sense proposed above. Will this mean that the phenomena under discussion – free will, reflective consciousness, phenomenal self – have been "understood"?

This depends on one's philosophy of consciousness. According to a *panpsychist* view, for instance, the answer would seem to be "yes," at least in a broad sense – the hyperset models presented would then constitute a demonstratively accurate model of the patterns in physical systems corresponding to the particular manifestations of universal experience under discussion. And it also seems that the answer would be "yes" according to a purely materialist perspective, since in that case we would have figured out what classes of physical conditions correspond to the "experiential reports" under discussion. Of course, both the panpsychist and materialist views are ones in which the "hard problem" is not an easy problem but rather a non-problem!

The ideas presented here have originated within a patternist perspective, in which what's important is to identify the patterns constituting a given phenomenon; and so we have sought to identify the patterns corresponding to free will, reflective consciousness and phenomenal self. The "hard problem" then has to do with the relationships between various qualities that these patterns are hypothesized to possess (experiential versus physical) ... but from the point of view of studying brains, building AI systems or conducting

our everyday lives, it is generally the patterns (and their subpatterns) that matter.

Finally, if the ideas presented above are accepted as a reasonable approach, there is certainly much more work to be done. There are many different states of consciousness, many different varieties of self, many different aspects to the experience of willing, and so forth. These different particulars may be modeled using hypersets, via extending and specializing the definitions proposed above. This suggested research program constitutes a novel variety of consciousness studies, using hypersets as a modeling language, which may be guided from a variety of directions including empirics and introspection.

C.7 Implications for Practical Work on Machine Consciousness

But what are the implications of the above ideas for *machine* consciousness in particular? One very clear implication is that digital computers probably can be just as conscious as humans can. Why the hedge "probably"? One reason is the possibility that there are some very odd, unanticipated restrictions on the patterns realizable in digital computers under the constraints of physical law. It is possible that special relativity and quantum theory, together, don't allow a digital computer to be smart enough to manifest self-reflective patterns of the complexity characteristic of human consciousness. (Special relativity means that big systems can't think as fast as small ones; quantum theory means that systems with small enough components have to be considered quantum computers rather than classical digital computers.) This seems extremely unlikely to me, but it can't be rated impossible at this point. And of course, even if it's true, it probably just means that machine consciousness needs to use quantum machines, or whatever other kind of machines the brain turns out to be.

Setting aside fairly remote possibilities, then, it seems that the patterns characterizing reflective consciousness, self and will can likely emerge from AI programs running on digital computers. But then, what more can be said about how these entities might emerge from the particular cognitive architectures and processes at play in the current AI field?

The answer to this question turns out to depend fairly sensitively on the particular AI architecture under consideration. Here we will briefly explore this issue in the context of CogPrime .

How do our hyperset models of reflective consciousness, self and will reflect themselves in the CogPrime architecture?

There is no simple answer to these questions, as CogPrime is a complex system with multiple interacting structures and dynamics, but we will give here a broad outline.

C.7.1 Attentional Focus in CogPrime

The key to understanding reflective consciousness in CogPrime is the ECAN (Economic Attention Networks) component, according to which each Atom in the system's memory has certain ShortTermImportance (STI) and LongTermImportance (LTI) values. These spread around the memory in a manner vaguely similar to activation spreading in a neural net, but using equations drawn from economics. The equations are specifically tuned so that, at any given time, a certain relatively small subset of Atoms will have significantly higher STI and LTI values than the rest. This set of important Atoms is called the AttentionalFocus, and represents the "moving bubble of attention" mentioned above, corresponding roughly to the Global Workspace in global workspace theory.

According to the patternist perspective, if some set of Atoms remains in the AttentionalFocus for a sustained period of time (which is what the ECAN equations are designed to encourage), then this Atom-set will be a persistent pattern in the system, hence a significant part of the system's mind and consciousness. Furthermore, the ECAN equations encourage the formation of densely connected networks of Atoms which are probabilistic attractors of ECAN dynamics, and which serve as hubs of larger, looser networks known as "maps." The relation between an attractor network in the AttentionalFocus and the other parts of corresponding maps that have lower STI, is conceptually related to the feeling humans have that the items in their focus of reflective consciousness are connected to other dimly-perceived items "on the fringes of consciousness."

The moving bubble of attention does not in itself constitute humanlike "reflective consciousness", but it prepares the context for this. Even a simplistic, animal-like CogPrime system with almost no declarative understanding of itself or ability to model itself, may still have intensely conscious patterns, in the sense of having persistent networks of Atoms frequently occupying its AttentionalFocus, its global workspace.

C.7.2 Maps and Focused Attention in CogPrime

The relation between focused attention and distributed cognitive maps in CogPrime bears some emphasis, and is a subtle point related to CogPrime knowledge representation, which takes both explicit and implicit forms. The explicit level consists of Atoms with clearly comprehensible meanings, whereas the implicit level consists of "maps" as mentioned above– collections of Atoms that become important in a coordinated manner, analogously to cell assemblies in an attractor neural net.

Formation of small maps seems to follow from the logic of focused attention, along with hierarchical maps of a certain nature. But the argument for

this is somewhat subtle, involving cognitive synergy between PLN inference and economic attention allocation.

The nature of PLN is that the effectiveness of reasoning is maximized by (among other strategies) minimizing the number of incorrect probabilistic independence assumptions. If reasoning on N nodes, the way to minimize independence assumptions is to use the full inclusion-exclusion formula to calculate interdependencies between the N nodes. This involves 2^N terms, one for each subset of the N nodes. Very rarely, in practical cases, will one have significant information about all these subsets. However, the nature of focused attention is that the system seeks to find out about as many of these subsets as possible, so as to be able to make the most accurate possible inferences, hence minimizing the use of unjustified independence assumptions. This implies that focused attention cannot hold too many items within it at one time, because if N is too big, then doing a decent sampling of the subsets of the N items is no longer realistic.

So, suppose that N items have been held within focused attention, meaning that a lot of predicates embodying combinations of N items have been constructed and evaluated and reasoned on. Then, during this extensive process of attentional focus, many of the N items will be useful in combination with each other - because of the existence of predicates joining the items. Hence, many HebbianLinks (Atoms representing statistical association relationships) will grow between the N items - causing the set of N items to form a map.

By this reasoning, focused attention in CogPrime is implicitly a map formation process – even though its immediate purpose is not map formation, but rather accurate inference (inference that minimizes independence assumptions by computing as many cross terms as is possible based on available direct and indirect evidence). Furthermore, it will encourage the formation of maps with a small number of elements in them (say, N<10). However, these elements may themselves be ConceptNodes grouping other nodes together, perhaps grouping together nodes that are involved in maps. In this way, one may see the formation of hierarchical maps, formed of clusters of clusters of clusters..., where each cluster has N<10 elements in it.

It is tempting to postulate that any intelligent system must display similar properties - so that focused consciousness, in general, has a strictly limited scope and causes the formation of maps that have *central cores* of roughly the same size as its scope. If this is indeed a general principle, it is an important one, because it tells you something about the general structure of concept networks associated with intelligent systems, based on the computational resource constraints of the systems. Furthermore this ties in with the architecture of the self.

C.7.3 Reflective Consciousness, Self and Will in CogPrime

So far we have observed the formation of simple maps in OpenCogPrime systems, but we haven't yet observed the emergence of the most important map: the self-map. According to the theory underlying CogPrime , however, we believe this will ensue once an OpenCogPrime -controlled virtual agent is provided with sufficiently rich experience, including diverse interactions with other agents.

The self-map is simply the network of Nodes and Links that a CogPrime system uses to predict, explain and simulate its own behavior. "Reflection" in the sense of cognitively reflecting on oneself, is modeled in CogPrime essentially as "doing PLN inference, together with other cognitive operations, in a manner heavily involving one's self-map."

The hyperset models of reflective consciousness and self presented above, appear in the context of CogPrime as approximative models of properties of maps that emerge in the system due to ECAN AttentionalFocus/map dynamics and its relationship with other cognitive processes such as inference. Our hypothesis is that, once a CogPrime system is exposed to the right sort of experience, it will internally evolve maps associated with reflective cognition and self, which possess an internal recursive structure that is effectively approximated using the hyperset models given above.

Will, then, emerges in CogPrime in part due to logical Atoms known as CausalImplicationLinks. A link of this sort is formed between A and B if the system finds it useful to hypothesis that "A causes B." If A is an action that the system itself can take (a GroundedSchemaNode, in CogPrime lingo) then this means roughly that "If I chose to do A, then B would be likely to ensue." If A is not an action the system can take, then the meaning may be interpreted similarly via abductive inference (i.e. via heuristic reasoning such as "If I *could* do A, and I did it, then B would likely ensue").

The self-map is a distributed network phenomenon in CogPrime 's AtomSpace, but the cognitive process called MapFormation may cause specific ConceptNodes to emerge that serve as hubs for this distributed network. These Self Nodes may then get CausalImplicationLinks pointing out from them – and in a mature CogPrime system, we hypothesize, these will correlate with the system's feeling of *willing*. The recursive structure of will emerges directly from the recursive structure of self, in this case – if the system ascribes cause to its self, then within itself there is also a model of its ascription of cause to its self (so that the causal ascription becomes part of the self that is being ascribed causal power), and so forth on multiple levels. Thus one has a finite-depth recursion that is approximatively modeled by the hyperset model of will described above.

All this goes well beyond what we have observed in the current CogPrime system (we have done some causal inference, but not yet in conjunction with

self-modeling), but it follows from the CogPrime design on a theoretical level, and we will be working over the next years to bring these abstract notions into practice.

C.7.4 Encouraging the Recognition of Self-Referential Structures in the AtomSpace

Finally, we consider the possibility that a CogPrime system might explicitly model its own self and behavior using hypersets.

This is quite an interesting possibility, because, according to the same logic as map formation: if these hyperset structures are explicitly recognized when they exist, they can then be reasoned on and otherwise further refined, which may then cause them to exist more definitively ... and hence to be explicitly recognized as yet more prominent patterns ... etc. The same virtuous cycle via which ongoing map recognition and encapsulation leads to concept formation, might potentially also be made to occur on the level of complex self-referential structures, leading to their refinement, development and ongoing complexity.

One relatively simple way to achieve this in CogPrime would be to encode hyperset structures and operators in the set of primitives of the "Combo" language that CogPrime uses to represent procedural knowledge (a simple LISP-like language with carefully crafted hooks into the AtomSpace and some other special properties). If this were done, one could then recognize self-referential patterns in the AtomTable via standard CogPrime methods like MOSES and PLN.

This is quite possible, but it brings up a number of other deep issues that go beyond the scope of this paper. For instance, most knowledge in CogPrime is uncertain, so if one is to use hypersets in Combo, one would like to be able to use them probabilistically. The most natural way to assign truth values to hyperset structure turns is to use infinite order probability distributions, as described in [Goe10b]. Infinite-order probability distributions are partially-ordered, and so one can compare the extent to which two different self-referential structures apply to a given body of data (e.g. an AtomTable), via comparing the infinite-order distributions that constitute their truth values. In this way, one can recognize self-referential patterns in an AtomTable, and carry out encapsulation of self-referential maps. This sounds very abstract and complicated, but the class of infinite-order distributions defined in the above-referenced papers actually have their truth values defined by simple matrix mathematics, so there is really nothing that abstruse involved in practice.

Clearly, with this subtle, currently unimplemented aspect of the CogPrime design we are veering rather far from anything the human brain could plausibly be doing in detail. This is fine, as CogPrime is not intended as a brain emulation. But yet, some meaningful connections may be drawn to neuro-

science. In **??** we have discussed how probabilistic logic might emerge from the brain, and also how the brain may embody self-referential structures like the ones considered here, via (perhaps using the hippocampus) encoding whole neural nets as inputs to other neural nets. Regarding infinite-order probabilities, it is certainly the case that the brain is wired to carry out matrix manipulations, and reduced infinite-order probabilities to them, so that it's not completely outlandish to posit the brain could be doing something mathematically analogous. Thus, all in all, it seems at least *plausible* that the brain could be doing something roughly analogous to what we've described here, though the details would obviously be very different.

C.8 Algebras of the Social Self

In the remainder of this appendix we will step even further out on our philosophico-mathematical limb, and explore the possibility that the recursive structures of the self involve mutual recursion according to the pattern of the quaternionic and octonionic algebras.

The argument presented in favor of this radical hypothesis has two steps. First, it is argued that much of human psychodynamics consists of "internal dialogue" between separate internal actors – some of which may be conceived as subselves a la [Row90], some of which may be "virtual others" intended to explicitly mirror other humans (or potentially other entities like animals or software programs). Second, it is argued that the structure of inter-observation among multiple inter-observing actors naturally leads to quaternionic and octonionic algebras. Specifically, the structure of inter-observation among three inter-observers is quaternionic; and the structure of inter-observation among four inter-observers is octonionic. This mapping between inter-observation and abstract algebra is made particularly vivid by the realization that the quaternions model the physical situation of three mirrors facing each other in a triangle; whereas the octonions model the physical situation of four mirrors facing each other in a tetrahedron, or more complex packing structures related to tetrahedra. Using these facts, we may phrase the main thesis to be pursued in the remainder of the appendix in a simple form: ***The structure of the self of an empathic social intelligence is that of a quaternionic or octonionic mirrorhouse.***

There is an intriguing potential tie-in with recent developments in neurobiology, which suggest that empathic modeling of other minds may be carried out in part via a "mirror neuron system" that enables a mind to experience another's actions, in a sense, "as if they were its own" [Ram06]. There are also echoes here of Buckminster Fuller's [FB82] philosophy, which viewed the tetrahedron as an essential structure for internal and external reality (since the tetrahedron is closely tied with the quaternion algebra).

C.9 The Intrinsic Sociality of the Self

We begin the next step of our journey with a theme that is generally neglected within AI yet is absolutely critical to humanlike intelligence: the **social** nature of the individual mind. In what sense may it be said that the self of an individual human being is a "social" system?

A 2001 special issue of "Journal of Consciousness Studies" [**?**] provided an excellent summary of recent research and thinking on this topic. A basic theme spanning several papers in the issue was as follows:

1. The human brain contains structures specifically configured to respond to other humans' behaviors (these appear to involve "mirror neurons" and associated "mirror neuron systems," on which we will elaborate below).
2. these structures are also used internally when no other people (or other agents) are present, because human self is founded on a process of continual interaction between "phenomenal self" and "virtual other(s)", where the virtual others are reflected by the same neural processes used to mirror actual others
3. so, the iteration between phenomenal self and actual others is highly wrapped up with the interaction between phenomenal self and virtual others

In other words, there is a complex dynamics according to which self is fundamentally grounded in sociality and social interactions. The social interactions that structure the self are in part grounded in the interactions between the brain structures generating the phenomenal self and the brain structures generating the virtual others. They are part of the dynamics of the self as well as part of the interactions between self and actual others. Human self is intrinsically not autonomous and independent, but rather is intrinsically dialogic and intersubjective.

Another way to phrase this is in terms of "empathy." That is, one can imagine an intelligence that attempted to understand other minds in a purely impersonal way, simply by reasoning about their behavior. But that doesn't seem to be the whole story of how humans do it. Rather, we do it, in part, by running simulations of the other minds internally – by spawning virtual actors, virtual selves within our own minds that emulate these other actors (according our own understanding). This is why we have the feeling of empathy – of feeling what another mind is feeling. It's because we actually *are* feeling what the other mind is feeling – in an approximation, because we're feeling what our internal simulation of the other mind is feeling. Thus, one way to define "empathy" is as the understanding of other minds via internal simulation of them. Clearly, internal simulation is not the only strategy the human mind takes to studying other minds – the patterns of errors we make in predicting others' behaviors indicates that there is also an inferential, analytical component to a human's understanding of others (Carruthers

and Smith, 1996). But empathic simulation is a key component, and we suggest that, in normal humans (autistic humans may be a counterexample; see Oberman et al, 2005), it is the most central aspect of other-modeling, the framework upon which other sorts of other-modeling such as inferencing are layered.

This perspective has some overlap with John Rowan's theory of human subpersonalities [Row90], according to which each person is analyzed as possessing multiple subselves representing different aspects of their nature appropriate to different situations. Subselves may possess different capabilities, sometimes different memories, and commonly differently biased views of the common memory store. Numerous references to this sort of "internal community" of the mind exist in literature, e.g. Proust's reference to "the several gentlemen of whom I consist."

Putting these various insights together, we arrive at a view of the interior of the human mind as consisting of not a single self but a handful of actors representing subselves and virtual others. In other words, we arrive at a perspective of human mind as social mind, not only in the sense that humans define themselves largely in terms of their interactions with others, but also in the sense that humans are substantially internally constituted by collections of interacting actors each with some level of self-understanding and autonomy. In the following sections we follow this general concept up by providing a specific hypothesis regarding the structure of this internal social mind: that it corresponds to the structures of certain physical constructs (mirrorhouses) and certain abstract algebras (quaternions, octonions and Clifford algebras).

C.10 Mirror Neurons and Associated Neural Systems

An increasingly popular line of thinking ties together the above ideas regarding self-as-social-system, with recent neurobiological results regarding the role of mirror neurons and associated neural systems in allowing human and animal minds to interpret, predict and empathize with other human and animal minds with which they interact. The biology of mirror neuron systems is still only partially understood, so that the tie-in between mirror neurons and psychological structures as discussed here must be viewed as highly subject to revision based on further refinement of our understanding in the biology of mirror neurons. Ultimately, the core AI and cognitive science ideas of this appendix would remain equally valid if one replaced "mirror neurons" and associated systems with some other, functionally similar neural mechanism. However, given that we do have some reasonably solid biological data – and some additional, associated detailed biological hypotheses – regarding the role of mirror neurons in supporting the functions of empathy and self, it is interesting to investigate what these data and hypotheses suggest.

In simplest terms, a mirror neuron is a neuron which fires both when an animal acts and when the animal observes the same action performed by another animal, especially one of the same species. Thus, the neuron is said to "mirror" the behavior of another animal – creating a similar neuronal activation patterns as if the observer itself were acting. Mirror neurons have been directly observed in primates, and are believed to exist in humans as well as in some other mammals and birds [Bla06]. Evidence suggestive of mirror neuron activity has been found in human premotor cortex and inferior parietal cortex. V.S. Ramachandran [Ram06] has been among the more vocal advocates of the important of mirror neurons, arguing that they may be one of the most important findings of neuroscience in the last decade, based on the likelihood of their playing a strong role in language acquisition via imitative learning.

The specific conditions under which mirror neuron activity occurs are still being investigated and are not fully understood. Among the classic examples probed in lab experiments are grasping behavior, and facial expressions indicating emotions such as disgust. When an ape sees another ape grasp something, or make a face indicating disgust, mirror neurons fire in the observing ape's brain, similar to what would happen if the observing ape were the one doing the grabbing or experiencing the disgust. This is a pretty powerful set of facts – what it says is that shared experience among differently embodied minds is not a purely cultural or psychological phenomenon, it's something that is wired into our physiology. We really can feel each others' feelings as if they were our own; to an extent, we may even be able to will each others' actions as if they were our own [**?**].

Equally interesting is that mirror neuron response often has to do with the perceived intention or goal of an action, rather than the specific physical action observed. If another animal is observed carrying out an action that is expected to lead to a certain goal, the observing animal may experience neural activity that it would experience if it had achieved this goal. Furthermore, mere visual observation of actions doesn't necessarily elicit mirror neuron activity. Recent studies [BBF+01, BLC+04] involved scanning the brains of various human subjects while they were observing various events, such another person speaking or biting something, a monkey lip-smacking or a dog barking. The mirror neurons were not activated by the sight of the barking dog – presumably because this was understood visually and not empathically (since people don't bark), but were activated by the sight of other people as well as of monkeys.

There is also evidence that mirror neurons may come to be associated with learned rather than just inherited capabilities. For instance, monkeys have mirror neurons corresponding to specific activities such as tearing paper, which are learned in the lab and have no close correlate in the wild [RC04].

C.10.1 Mirror Systems

Perhaps the most ambitious hypothesis regarding the role of mirror neurons in cognition is Rizzolatti and Arbib's [RA98] Mirror System Hypothesis, which conjectures that neural assemblies reliant on mirror neurons played a key role in the evolution of language. These authors suggest that Broca's area (associated with speech production) evolved on top of a mirror system specialized for grasping, and inherited from this mirror system a robust capacity for pattern recognition and generation, which was then used to enable imitation of vocalizations, and to encourage "parity" in which associations involving vocalizations are roughly the same for the speaker as for the hearer. According to the MSH, the evolution of language proceeded according to the following series of steps [ABR06]:

1. S1: Grasping.
2. S2: A mirror system for grasping, shared with the common ancestor of human and monkey.
3. S3: A system for simple imitation of grasping shared with the common ancestor of human and chimpanzee. The next 3 stages distinguish the hominid line from that of the great apes:
4. S4: A complex imitation system for grasping.
5. S5: Protosign, a manual-based communication system that involves the breakthrough from employing manual actions for praxis to using them for pantomime (not just of manual actions), and then going beyond pantomime to add conventionalized gestures that can disambiguate pantomimes.
6. S6: Protospeech, resulting from linking the mechanisms for mediating the semantics of protosign to a vocal apparatus of increasing flexibility. The hypothesis is not that S5 was completed before the inception of S6, but rather that protosign and protospeech evolved together in an expanding spiral.
7. S7: Language: the change from action-object frames to verb-argument structures to syntax and semantics.

As we will point out below, one may correlate this series of stages with a series of mirrorhouses involving an increasing number of mirrors. This leads to an elaboration of the MSH, which posits that evolutionarily, as the mirrorhouse of self and attention gained more mirrors, the capability for linguistic interaction became progressively more complex.

C.11 Quaternions and Octonions

In this section, as a preparation for our mathematical treatment of mirrorhouses and the self, we review the basics of the quaternion and octonion

algebras. This is not original material, but it is repeated here because it is not well known outside the mathematics and physics community. Readers who want to learn more should follow the references.

Most readers will be aware of the real numbers and the complex numbers. The complex numbers are formed by positing an "imaginary number" i so that i*i=-1, and then looking at "complex numbers" of the form a+bi, where a and b are real numbers. What is less well known is that this approach to extending the real number system may be generalized further. The quaternions are formed by positing three imaginary numbers i, j and k with i*i=j*j=k*k=-1, and then looking at "quaternionic numbers" of the form a + bi + cj + dk. The octonions are formed similarly, by positing 7 imaginary numbers i,j,k,E,I,J,K and looking at "octonionic numbers" defined as linear combinations thereof.

Why 3 and 7? This is where the math gets interesting. The trick is that only for these dimensionalities can one define a multiplication table for the multiple imaginaries so that unique division and length measurement (norming) will work. For quaternions, the "magic multiplication table" looks like

$$i * j = kj * i = -k$$

$$j * k = ik * j = -i$$

$$k * i = ji * k = -j$$

Using this multiplication table, for any two quaternionic numbers A and B, the equation

$$x * A = B$$

has a unique solution when solved for x. Quaternions are not commutative under multiplication, unlike real and complex numbers: this can be seen from the above multiplication table in which e.g. i*j is not equal to j*i. However, quaternions are normed: one can define ||A||for a quaternion A, in the familiar root-mean-square manner, and get a valid measure of length fulfilling the mathematical axioms for a norm.

Note that you can also define an opposite multiplication for quaternions: from i*j = k you can reverse to get j*i = k, which is an opposite multiplication, that still works, and basically just constitutes a relabeling of the quaternions. This is different from the complex numbers, where there is only one workable way to define multiplication.

The quaternion algebra is fairly well known due to its uses in classical physics and computer graphics (Hanson, 2006); the octonion algebra, also known as Cayley's octaves, is less well known but is adeptly reviewed by John Baez (2002).

The magic multiplication table for 7 imaginaries that leads to the properties of unique division and normed-ness is as follows:

1	**i**	**j**	**k**	**E**	**I**	**J**	**K**
i	-1	k	-j	I	-E	-K	J
j	-k	-1	i	J	K	-E	-I
k	j	-i	-1	K	-J	I	-E
E	-I	-J	-K	-1	i	j	k
I	E	-K	J	-i	-1	-k	j
J	K	E	-I	-j	k	-1	-i
K	-J	I	E	-k	-j	i	-1

Actually this is just one of 480 basically equivalent (and equally "magical") forms of the octonionic multiplication table (as opposed to the 2 varieties for quaternions, mentioned above). Note that, according to this or any of the other 479 tables, octonionic multiplication is neither commutative nor associative; but octonions do satisfy a weaker form of associativity called alternativity, which means that the subalgebra generated by any two elements is associative.

As it happens, the only normed division algebras over the reals are the real, complex, quaternionic and octonionic number systems. These four algebras also form the only alternative, finite-dimensional division algebras over the reals. These theorems are nontrivial to prove, and fascinating to contemplate – and even more fascinating when one considers their possible connection to the emergent structures of general intelligence.

C.12 Modeling Mirrorhouses Using Quaternions and Octonions

Now let's move from algebras to mirrors – houses of mirrors, to be precise. Interestingly shaped houses of mirrors!

Mirrorhouses are structures built up?from mutually facing mirrors which reflect each others' reflections. The?simplest mirrorhouse possible to construct is made of two facing?mirrors, X and Y. X reflects Y and Y reflects X.

In terms of hypersets, a simple 2-mirror mirrorhouse may be crudely described as:

$$X = \{Y\}$$
$$Y = \{X\}$$

(ignoring the inversion effect of mirroring).

Note that if we try to unravel this hyperset by inserting one element into the other we arrive at an infinite regress:

$$Y = \{X = \{Y = \{X = \{Y = \{X = \{Y = \{\{X = \{Y = \{...\}\}\}\}\}\}\}\}\}\}$$

This corresponds to the illusory infinite tube which interpenetrates both mirrors.

Suppose now that we constructed a mirrorhouse from *three* mirrors instead of two. What hyper-structure would this have? Amazingly it turns out that it has precisely the structure of the quaternion imaginaries.

Let i, j and k be hypersets representing three facing mirrors. We then have that

$$i = \{j, k\}$$

$$j = \{k, i\}$$

and

$$k = \{i, j\}$$

where the notation i={j,k} means, e.g. that mirror i reflects mirrors j and k in that order.

With three mirrors ordering now starts playing a vital role because mirroring inverts left/right-handedness. If we denote the mirror?inversion operation by "-" we have that

$$i = \{j, k\} = -\{k, j\}$$

$$j = \{k, i\} = -\{i, k\}$$

$$k = \{i, j\} = -\{j, i\}$$

But the above is exactly the structure of the quaternion triple of imaginaries:

$$i = j * k = -k * j$$

$$j = k * i = -i * k$$

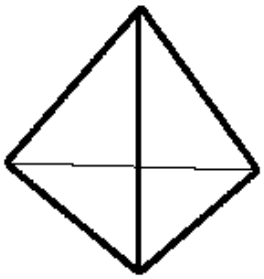
$$k = i * j = -j * i$$

The quaternion algebra therefore is the precise model of three facing mirrors, where we see mirror inversion as the quaternionic anti-commutation. The two versions of the quaternion multiplication table correspond to the two possible ways of arranging three mirrors into a triangular mirrorhouse.

When we move on to octonions, things get considerably subtler – though no less elegant, and no less conceptually satisfying. While there are 2 possible quaternionic mirrorhouses, there are 480 possible octonionic mirrorhouses, corresponding to the 480 possible variant octonion multiplication tables!

Recall that the octonions have 7 imaginaries i,j,k,E,I,J,K, which have 3 algebraic generators i,j,E (meaning that combining these three imaginaries can give rise to all the others). The third generator E is distinguished from the others, and we can vary it to get the 480 multiplications/mirrorhouses.

The simplest octonionic mirrorhouse is simply the tetrahedron (see Figure C.12). More complex octonionic mirrorhouses correspond to tetrahedra with extra mirrors placed over their internal corners, as shown in Figure C.12. This gives rise to very interesting geometric structures, which have been explored by Buckminster Fuller and also by various others throughout history.



Start with a 3-dimensional tetrahedron of 4 facing mirrors. Let the floor be the distinguished third generator E and the 3 walls be I,J,K (with a specific assignment of walls to imaginaries, of course). Then, by reflection through the E floor, the reflected I J K become i j k, and we now have all 7 imaginary octonions. This relatively simple tetrahedral mirrorhouse corresponds to one of the 480 different multiplications; the one given in the table above.

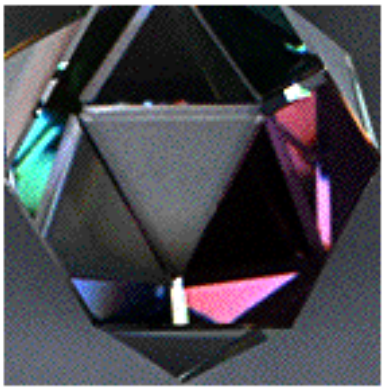
To get another we truncate the tetrahedron. Truncation puts a mirror parallel to the floor, making a mirror roof. Then, when you look up at the mirror roof, you see the triangle roof parallel to the floor E. The triangle roof parallel to the floor E represents the octonion -E, and reflection in the roof -E gives 7 imaginary octonions with the multiplication rule in which -E is the distinguished third generator.

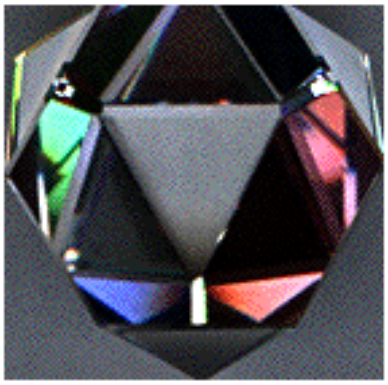
Looking up from the floor, you will also see 3 new triangles having a common side with the triangle roof -E, and 6 new triangles having a common vertex with the triangle roof -E.

The triangle roof + 9 triangles = 10 triangles form half of the faces (one hemisphere) of a 20-face quasi-icosahedron. The quasi-icosahedron is only qualitatively an icosahedron, and is not exact, since the internal angle of the pentagonal vertex figure of the reflected quasi-icosahedron is not 108 degrees, but is 109.47 degrees (the octahedral dihedral angle), and the vertex angle is not 72 degrees, but is 70.53 degrees (the tetrahedral dihedral angle). (To get an exact icosahedral kaleidoscope, three of the triangles of the tetrahedron should be golden isosceles triangles.)

Each of the 9 new triangles is a "reflection roof" defining another multiplication. Now, look down at the floor E to see 9 new triangles reflected from the 9 triangles adjoining the roof -E. Each of these 9 new triangles is a "reflection floor" defining another multiplication. We have now 1 + 1 + 9 + 9 = 20 of the 480 multiplications.

Just as we put a roof parallel to the floor E by truncating the top of the tetrahedral pyramid, we can put in 3 walls parallel to each of the 3 walls I, J, K by truncating the other 3 points of the tetrahedron, thus getting 3x20 = 60 more multiplications. That gives us 20 + 60 = 80 of the 480 multiplications.





To get the rest, recall that we fixed the walls I, J, K in a particular order with respcet to the floor E. There are 3! = 6 permutations of the walls I, J, K Taking them into account, we get all 6x80 = 480 multiplications.

In mathematical terms, this approach effectively fixes the 20-face quasi-icosahedron and varies the 4 faces of the EIJK tetrahedron according to the 24-element binary tetrahedral group {3,3,2} = SL(2,3) to get the 20x24 = 480 multiplications.

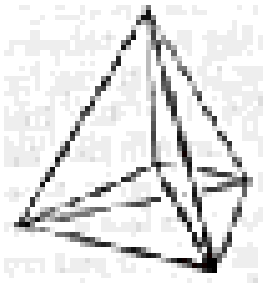
Note that the truncated tetrahedron with a quasi-icosahedron at each vertex combines two types of symmetries:

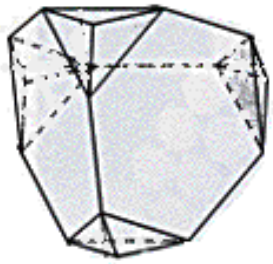
1. **tetrahedral**, related to the square and the square root of 2, which gives open systems like: an arithmetic series overtone acoustic musical scale with common difference 1/8; the Roman Sacred Cut in architecture; and multilayer space-filling cuboctahedral crystal growth.
2. **icosahedral**, related to the pentagon, the Golden Mean (aka Golden Section), and Fibonacci sequences, which gives closed systems like: a harmonic pentatonic musical scale; Le Corbusier's Modulor; and single-layer icosahedral crystals.

It is interesting to observe that the binary icosahedral group is isomorphic to the binary symmetry group of the 4-simplex, which may be called the

pentahedron and which David Finkelstein and Ernesto Rodriguez (1984) have called the "Quantum Pentacle." A pentahedron has 5 vertices, 10 edges, 10 areas, and 5 cells. The 10 areas of a pentahedron correspond to the 10 area faces of one hemisphere of an icosahedron.

The pentahedron projected into 3 dimensions looks like a tetrahedron divided into 4 quarter-tetrahedra (Figure C.12). If you add a quarter-tetrahedron to each truncation of a truncated tetrahedron, you get a space-filling polytope (Figure C.12) that can be centered on a vertex of a 3-dimensional diamond packing to form a Dirichlet domain of the 3-dimensional diamond packing (Figure **??**). (A Dirichlet domain of a vertex in a packing is the set of points in the space in which the packing is embedded that are nearer to the given vertex than to any other.) The 4 most distant vertices of the Dirichlet domain polytope are vertices of the dual diamond packing in 3-dimensional space.

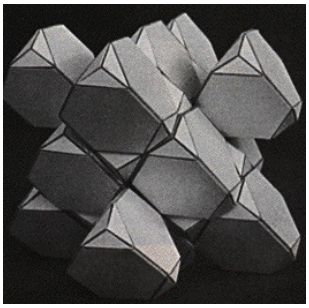




All in all, we conclude that:

1. In its simplest form the octonion mirrorhouse is a tetrahedral mirrorhouse
2. In its more general form, the octonion mirrorhouse shows a tetrahedral diamond packing network of quasi-icosahedra, or equivalently, of quasi-pentahedra

Observation as Mirroring



Now we proceed to draw together the threads of the previous sections: mirror neurons and subselves, mirrorhouses and normed division algebras.

To map the community of actors inside an individual self into the mirrorhouse/algebraic framework of the previous section, it suffices to interpret the above

$$X = \{Y\}$$

$$Y = \{X\}$$

as

"X observes Y"
"Y observes X"

(e.g. we may have X= primary subself, Y=inner virtual other), and the above

$$i = \{j, k\}$$

$$j = \{k, i\}$$

$$k = \{i, j\}$$

as

"i observes {j observing k}"
"j observes {k observing i}"
"k observes {i observing j}"

Then we can define the - observation as an inverter of observer and observed, so that e.g.

$$\{j, k\} = -\{k, j\}$$

We then obtain the quaternions

$$i = j * k = -k * j$$

$$j = k * i = -i * k$$

$$k = i * j = -j * i$$

where multiplication is observation and negation is reversal of the order of observation. Three inter-observers = quaternions.

The next step is mathematically natural: if there are four symmetric inter-observers, one obtains the octonions, according to the logic of the above-described tetrahedral/tetrahedral-diamond-packing mirrorhouse. Octonions may also be used to model various situations involving more than four observes with particular asymmetries among the observers (the additional observers are the corner-mirrors truncating the tetrahedron.)

Why not go further? Who's to say that the internal structure of a social mind isn't related to mirrorhouses obtained from more complex shapes than tetrahedra and truncated tetrahedra? This is indeed not impossible, but intuitively, we venture the hypothesis that where human psychology is concerned, the octonionic structure is complex enough. Going beyond this level one loses the normed division-algebra structure that makes the octonions a reasonably nice algebra, and one also gets into a domain of dramatically escalated combinatorial complexity.

Biologically, what this suggests is that the MSH of Rizzolatti and Arbib just scratches the surface. The system of mirror neurons in the human mind may in fact be a "mirrorhouse system," involving four different cell assemblies, each involving substantial numbers of mirror neurons, and arranged in such a manner as to recursively reflect and model one another. This is a concrete neurological hypothesis which is neither strongly suggested nor in any way refuted by available biological data: the experimental tools at our current disposal are simply not adequate to allow empirical exploration of this sort of hypothesis. The empirical investigation of cell assembly activity is possible now only in a very primitive way, using crude tools such as voltage-sensitive dyes which provide data with a very high noise level (see e.g. [CMH+07]. Fortunately though, the accuracy of neural measurement technology is increasing at an exponential rate [Kur06], so there is reason to believe that within a few decades hypotheses such as the presently positive "neural mirrorhouse" will reside in the domain of concretely-explorable rather than primarily-theoretical science.

And finally, we may take this conceptual vision one more natural step. The mirrorhouse inside an individual person's mind is just one small portion of the overall social mirrorworld. What we really have is a collection of interlocking mirrorhouses. If one face of the tetrahedron comprising my internal mirrorhouse at a certain moment corresponds to one of your currently active

subselves, then we may view our two selves at that moment as two adjacent tetrahedra. We thus arrive at a view of a community of interacting individuals as a tiling of part of space using tetrahedra, a vision that would have pleased Buckminster Fuller very much indeed.

C.13 Specific Instances of Mental Mirrorhousing

We've voyaged fairly far out into mathematical modeling land – what does all this mean in terms of our everyday lives, or in terms of concrete AGI design?

Most examples of mental mirrorhousing, I suggest, are difficult for us to distinguish introspectively from other aspects of our inner lives. Mirroring among multiple subselves, simulations of others and so forth is so fully woven into our consciousness that we don't readily distinguish it from the rest of our inner life. Because of this, the nature of mental mirroring is most easily understood via reference to "extreme cases."

For instance, consider the following rather mundane real-life situation: Ben needs to estimate the time-duration of a software project that has been proposed for the consulting division of his AI software company. Ben knows he typically underestimates the amount of time required for a project, but that he can usually arrive at a more accurate estimate via conversation with his colleague Cassio. But Cassio isn't available at the moment; or Ben doesn't want to bother him. So, Ben simulates an "internal Cassio," and they dialogue together, inside Ben's "mind's eye." This is a mirror facing a mirror – an internal Ben mirroring an internal Cassio.

But this process in itself may be more or less effective depending on the specifics –depending on, for example, which aspects of Ben or Cassio are simulated. So, an additional internal observing mind may be useful for, effectively, observing multiple runs of the "Ben and Cassio conversation simulator" and studying and tuning the behavior. Now we have a quaternionic mirrorhouse.

But is there a deeper inner observer watching over all this? In this case we have an octonionic, tetrahedral mirrorhouse.

The above is a particularly explicit example – but we suggest that much of everyday life experience consists of similar phenomena, where the different inter-mirroring agents are not necessarily associated with particular names or external physical agents, and thus are more difficult to tangibly discussed. As noted above, this relates closely to Rowan's analysis of human personality as consisting largely of the interactional dynamics of various never-explicitly-articulated and usually-not-fully-distinct subpersonalities.

For another sort of example, consider the act of creativity, which in (Goertzel, 1997) is modeled in terms of a "creative subself": a portion of the mind that is specifically devoted to creative activity in one more more media, and has its own life and awareness and memory apart from the primary self-structure. The creative subself may create a work, and present it to the

main subself for consideration. The three of these participants – the primary subself, the creative subself and the creative work – may stand in a relationship of quaternionic mirroring. And then the meta-self who observes this threefold interaction completes the tetrahedral mirrorhouse.

Next, let us briefly consider the classic Freudian model of personality and motivation. According to Freud (1962), much of our psychology consists of interaction between ego, superego and id. Rather than seeking to map the precise Freudian notions into the present framework, we will briefly comment on how ideas inspired by these Freudian notions might play a role in the present framework. The basic idea is that, to the extent that there are neuropsychological subsystems corresponding to Freudian ego, superego and id, these subsystems may be viewed as agents that mirror each other, and hence as a totality may be viewed as a quaternionic mirrorhouse. More specifically we may correlate

1. **ego** with the neuropsychological structure that Thomas Metzinger (2004) has identified as the "phenomenal self"
2. **superego** with the neuropsychological structure that represents the mind's *learned goal system* – the set of goals that the system has created
3. **id** with the neuropsychological structure that represents the mind's *inbuilt goal system*, which largely consists of basic biological drives

Using this interpretation, we find that a quaternionic ego/superego/id mirrorhouse may indeed play a role in human psychology and cognition. However, there is nothing in the theoretical framework being pursued here to suggest that this particular configuration of inter-observers has the foundational significance Freud ascribed to it. Rather, from the present perspective, this Freudian triarchy appears as important configuration (but not the only one) that may arise within the mirrorhouse of focused attention.

And, of course, if we add in the internal observing eye that allowed Freud to identify this system in the first place, and we have an octonionic, tetrahedral mirrorhouse.

Finally, let us consider the subjective experience of meditation, as discussed e.g. in [Aus99]. Here we have "consciousness without an object" [MW83], which may be understood as the infusion of the mental mirrorhouse with attention but not content. Each mirror is reflecting the others, without any image to reflect except the mirrors themselves.

C.14 Mirroring in Development

Another naturally arising question regards the origin of the mental mirrorhouse faculty, both evolutionarily and developmentally. In both cases, the

obvious hypothesis is that during the course of growth, the inner mirrorhouse gains the capability for using more and more mirrors. First comes the capability to internally mirror an external agent; then comes the capability to internally encapsulate an inter-observation process; then comes the capability to internally observe an inter-observation process; then comes the capability to internally observe the observation of an inter-observation process. Of course, the hierarchy need not terminate with the octonionic mirrorhouse; but qualitatively, our suggestion is that levels beyond the octonionic may generally beyond the scope of what the human brain/mind needs to deal with given its limited environment and computational processing power.

To get a better grip on this posited growth process, let us return to Rizzolatti and Arbib's hypothesized role for mirror neurons in language learning. Their stage S5, as described above, involves "proto-signs," which may have initially consisted of pantomime used indirectly (i.e. used, not necessarily to denote specific motor actions, but to denote other activities loosely resembling those motor actions). A mental mirrorhouse corresponding to proto-signs may be understood to comprise 3 mirrors

1. Observer
2. Pantomimer (carrying out manual actions)
3. Object of pantomime (carrying out non-manual actions)

The hypothesis becomes that, via recollecting instances of pantomime using a quaternionic mirrorhouse, the mind imprints pantomimes on the long-term memory, so that they become part of the unconscious in a manner suitable to encourage the formation of new and richer pantomimes.

In general, going beyond the particular example of pantomime, we may posit a quaternionic mirrorhouse corresponding to

1. Observer
2. Symbols
3. Referent

The addition of a fourth mirror then corresponds to ***reflection on the process of symbolization***, which is not necessary for use of language but is necessary for conscious creation of language, as is involved for instance in formalizing grammar or creating abstract mathematics.

There is a clear and fascinating connection here with Piagetan developmental psychology, as reviewed in Chapter 11 above, in which the capability for symbolization is posited to come along with the "concrete operational" stage of development (between ages 7-14 in the average child); and the capability for abstract formal reasoning comes later in the "formal" stage of development. The natural hypothesis in this connection is that the child's mind during the concrete operational stage possesses only a quaternionic mirrorhouse (or at least, that only the quaternionic mirrorhouse is highly functional at this stage); and that the advent of the formal stage corresponds to the advent of the octonionic mirrorhouse.

This hypothesis has interesting biological applications, in the context of the previously hypothesized relationship between mirror neurons and mental mirroring. In this case, if the hypothesized correspondence between number-of-mirrors and developmental stages exists, then it should eventually be neurologically observable via studying the patterns of interaction of cell assemblies whose dynamics are dominated by mirror neurons, in the brains of children at different stages of cognitive development. As noted above, however, experimental neuroscience is currently nowhere near being able to validate or refute such hypotheses, so we must wait at least a couple decades before pursuing this sort of empirical investigation.

C.15 Concluding Remarks

Overall, the path traced in this appendix has been a somewhat complex one, but the broad outline of the story is summarizable compactly.

Firstly, there may well be elegant recursive, self-referential structures underlying reflective consciousness, will and self.

And secondly, there may plausibly be elegant abstract-algebraic symmetries lurking within the social substructures of the self. The notion of "emergent structures of mind" may include emergent *algebraic* structures arising via the intrinsic algebra of reflective processes.

We have some even more elaborate and speculative conjectures extending the ideas given here, but will not burden the reader with them – we have gone as far as we have here, largely to indicate the sort of ideas that arise when one takes the notion of emergent mind structures seriously.

Ultimately, abstract as they are, these ideas must be pursued empirically rather than via conceptual argumentation and speculation. If the CogPrime engineering programme is successful, the emergence or otherwise of the structures discussed here, and others extending them, will be discoverable via the mundane work of analyzing system logs.

Appendix D
GOLEM: Toward an AGI Meta-Architecture Enabling Both Goal Preservation and Radical Self-Improvement

D.1 Introduction

One question that looms large when thinking about the ultimate roadmap for AGI and the potential for self-modifying AGI systems is: How to create an AGI system that will **maintain some meaningful variant of its initial goals even as it dramatically revises and improves itself** – and as it becomes so much smarter via this ongoing improvement that in many ways it becomes incomprehensible to its creators or its initial condition. We would like to be able to design AGI systems that are massively intelligent, creatively self-improving, probably beneficial, and almost surely not destructive.

At this point, it's not terribly clear whether an advanced CogPrime system would have this desirable property or not. It's certainly not *implausible* that it would, since CogPrime does have a rich explicit goal system and is oriented to spend a significant percentage of its effort rationally pursuing its goals. And with its facility for reinforcement and imitation learning, CogPrime is well suited to learn ethical habits from its human teachers. But all this falls very far short of any kind of guarantee.

In this appendix we'll outline a general AGI meta-architecture called GOLEM (the Goal Oriented LEarning Meta-architecture), that can be used as a "wrapper" for more detailed AGI architectures like CogPrime , and that appears (but hasn't been formally proved) to have more clearly desirable properties in terms of long-term ethical behavior. From a CogPrime perspective, GOLEM may be viewed as a specific CogPrime configuration, which has powerful "AGI safety" properties but also demands a lot more computational resources than many other CogPrime configurations would.

To specify these notions a bit further, we may define an intelligent system as *steadfast* if, over a long period of time, it *either continues to pursue the same goals it had at the start of the time period, or stops acting altogether.* In this terminology, one way to confront the problem of creating probably-

beneficial, almost surely non-destructive AGI, is to solve the two problems of:

- How to encapsulate the goal of beneficialness in an AGI's goal system
- How to create steadfast AGI, in a way that applies to the "beneficialness" goal among others

Of course, the easiest way to achieve steadfastness is to create a system that doesn't change or grow much. And the interesting question raised is how to couple steadfastness with ongoing, radical, transformative learning.

In this appendix we'll present a careful semi-formal argument that, under certain reasonable assumptions (and given a large, but not clearly long-term infeasible amount of computer power), the GOLEM meta-architecture is likely to be both steadfast and massively, self-improvingly intelligent. Full formalization of the argument is left for later, and may be a difficult task even if the argument is correct.

An alternate version of GOLEM is also described, which possesses more flexibility to adapt to an unknown future, but lacks a firm guarantee of steadfastness.

Discussion of the highly nontrivial problem of "how to encapsulate the goal of beneficialness in an AGI's goal system" is also left for elsewhere (see [Goe10a] for some informal discussion). As reviewed already in Chapter 12 we suspect this will substantially be a matter of interaction and education rather than mainly a matter of explicitly formulating ethical content and telling or feeding it to an AGI system.

D.2 The Goal Oriented Learning Meta-Architecture

The Goal Oriented LEarning Meta-architecture (GOLEM) refers to an AGI system S with the following high-level meta-architecture, depicted roughly in Figure D.1:

- **Goal Evaluator** = component that calculates, for each possible future world (including environment states and internal program states), how well this world fulfills the goal (i.e. it calculates the "utility" of the possible world)
  - it may be that the knowledge supplied to the GoalEvaluator initially (the "base GEOP" i.e. "base GoalEvaluator Operating Program") is not sufficient to determine the goal-satisfaction provided by a world-state; in that case the GoalEvaluator may produce a probability distribution over possible goal-satisfaction values
  - initially the GoalEvaluator may be supplied with an inefficient algorithm encapsulating the intended goals, which may then be optimized

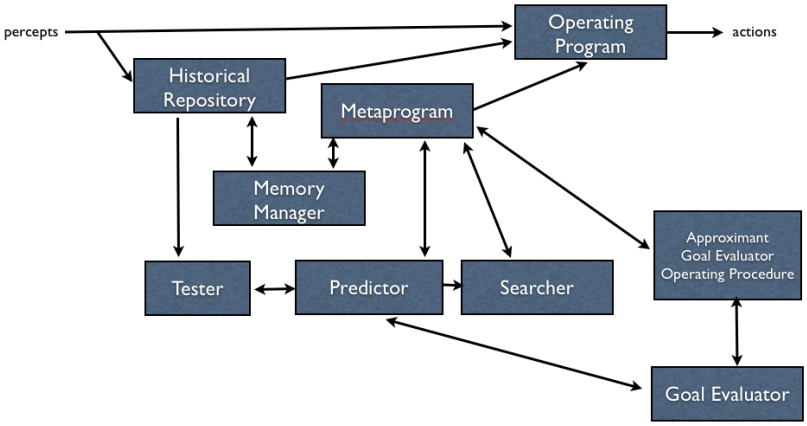


Fig. D.1 The GOLEM meta-architecture. Single-pointed errors indication information flow; double-pointed arrows indicate more complex interrelationships.

and approximated by application of the Searcher (thus leading to a GEOP different from the base GEOP)
  - if the GoalEvaluator uses a GEOP produced by the Searcher, then there may be an additional source of uncertainty involved, which may be modeled by having the GoalEvaluator output a second-order probability distribution (a distribution over distributions over utility values), or else by collapsing this into a first-order distribution

- **HistoricalRepository** = database storing the past history of S's internal states and actions, as well as information about the environment during S's past
- **Operating Program** = the program that S is governing its actions by, at a given point in time
  - chosen by the Metaprogram as the best program the Searcher has found, where "best" is judged as "highest probability of goal achievement" based on the output of the Predictor and the Goal Evaluator
- **Predictor** = program that estimates, given a candidate operating program P and a possible future world W, the odds of P leading to W
- **Searcher** = program that searches through program space to find a new program optimizing a provided objective function
- **Memory Manager program** = program that decides when to store new observations and actions in the Historical Repository, and which ones to delete in order to do so; potentially it may be given some hard-wired constraints to follow, such as "never forget human history, or the previous century of your life."

- **Tester** = hard-wired program that estimates the quality of a candidate Predictor, using a simple backtesting methodology
  - That is, the Tester assesses how well a Predictor would have performed in the past, using the data in the HistoricalRepository
- **Metaprogram** = fixed program that uses Searcher program to find a good
  - Searcher program (judged by the quality of the programs it finds, as judged by the Predictor program)
  - Predictor program (as judged by the Tester's assessments of its predictions)
  - Operating Program (judged by Predictor working with Goal Evaluator, according to the idea of choosing an Operating Program with the maximum expected goal achievement)
  - GoalEvaluator Operating Program (judged by the Tester, evaluating whether a candidate program effectively predicts goal-satisfaction given program-executions, according to the HistoricalRepository)
  - Memory Manager (as judged by Searcher, which rates potential memory management strategies based on the Predictor's predictions of how well the system will fare under each one)

  The Metaprogram's choice of Operating Program, Goal Evaluator Operating Program and Memory Manager may all be interdependent, as the viability of a candidate program for each of these roles may depend on what program is playing each of the other roles. The metaprogram also determines the amount of resources to allocate to searching for a Searcher versus a Predictor versus an OP, according to a fixed algorithm for parameter adaptation.

While this is a very abstract "meta-architecture", it's worth noting that it could be implemented using CogPrime or any other practical AGI architecture as a foundation – in this case, CogPrime is "merely" the initial condition for the OP, the Memory Manager, the Predictor and the Searcher. However, demonstrating that self-improvement can proceed at a useful rate in any particular case like this, may be challenging.

Note that there are several fixed aspects in the above: the MetaProgram, the Tester, the GoalEvaluator, and the structure of the HistoricalRepository. The standard GOLEM, with these aspects fixed, will also be called the *fixed GOLEM*, in contrast to an *adaptive GOLEM* in which *everything* is allowed to be adapted based on experience.

D.2.1 Optimizing the GoalEvaluator

Note that the GoalEvaluator may need to be very smart indeed to do its job. However, an important idea of the architecture is that the optimization of the GoalEvaluator's functionality may be carried out as part of the system's overall learning *.

In its initial and simplest form, the GoalEvaluator's internal Operating Program (GEOP) could basically be a giant simulation engine, that tells you, based on a codified definition of the goal function: in world-state W, the probability distribution of goal-satisfaction values is as follows. It could also operate in various other ways, e.g. by requesting human input when it gets confused in evaluating the desirability of a certain hypothetical world-state; by doing similarity matching according to a certain codified distance measure against a set of desirable world-states; etc.

However, the Metaprogram may supplement the initial "base GEOP" with an intelligent GEOP, which is learned by the Searcher, after the Searcher is given the goal of finding a program that will

- accurately agree with the base GEOP across the situations in the HistoricalRepository, as determined by the Tester
- be as compact as possible

In this approach, there is a "base goal evaluator" that may use simplistic methods, but then the system learns programs that do approximately the same thing as this but perhaps faster and more compactly, and potentially *embodying more abstraction*. Since this program learning has the specific goal of learning efficient approximations to what the GoalEvaluator does, it's not susceptible to "cheating" in which the system revises its goals to make them easier to achieve (unless the whole architecture gets broken).

What is particularly interesting about this mechanism is: it provides a built-in mechanism for extrapolation beyond the situations for which the base GEOP was created. The Tester requires that the learned GEOPs must agree with the base GEOP on the HistoricalRepository, but for cases not considered in the HistoricalRepository, the Metaprogram is then doing Occam's Razor based program learning, seeing a compact and hence rationally generalizable explanation of the base GEOP.

D.2.2 Conservative Meta-Architecture Preservation

Next, the GOLEM meta-architecture assumes that the goal embodied by the GoalEvaluator includes, as a subgoal, the preservation of the overall meta-

* this general idea was introduced by Abram Demski upon reading an earlier draft of this article, though he may not agree with the particular way I have improvised on his idea here

architecture described above (with a fallback to inaction if this seems infeasible). This may seem a nebulous assumption, but it's not hard to specify if one thinks about it the right way.

For instance, one can envision each of the items in the above component list as occupying a separate hardware component, with messaging protocols established for communicating between the components along cables. Each hardware component can be assumed to contain some control code, which is connected to the I/O system of the component and also to the rest of the component's memory and processors.

Then what we must assume is that the goal includes the following criteria, which we'll call *conservative meta-architecture preservation*:

1. No changes to the hardware or control code should be made except in accordance with the second criterion
2. If changes to the hardware or control code are found, then the system should stop acting (which may be done in a variety of ways, ranging from turning off the power to self-destruction; we'll leave that unspecified for the time being as that's not central to the point we want to make here)

Any world-state that violates these criteria, should be rated extremely low by the GoalEvaluator.

D.3 The Argument For GOLEM's Steadfastness

Our main goal here is to argue that a program with (fixed) GOLEM meta-architecture will be steadfast, in the sense that it will maintain its architecture (or else stop acting) while seeking to maximize the goal function implicit in its GoalEvaluator.

Why do we believe GOLEM can be steadfast? The basic argument, put simply, is that: If

- the GoalEvaluator and environment together have the property that:
  - world-states involving conservative meta-architecture preservation tend to have very high fitness
  - world-states not involving conservative meta-architecture preservation tend to have very low fitness
  - world-states approximately involving conservative meta-architecture preservation tend to have intermediate fitness
- the initial Operating Program has a high probability of leading to world-states involving conservative meta-architecture preservation (and this is recognized by the GoalEvaluator)

then the GOLEM meta-architecture will be preserved. Because: according to the nature of the metaprogram, it will only replace the initial Operating

Program with another program that is predicted to be *more* effective at achieving the goal, which means that it will be unlikely to replace the current OP with one that doesn't involve conservative meta-architecture preservation.

Obviously, this approach doesn't allow full self-modification; it assumes certain key parts of the AGI (meta)architecture are hard-wired. But the hard-wired parts are quite basic and leave a lot of flexibility. So the argument covers a fairly broad and interesting class of goal functions.

D.4 A Partial Formalization of the Architecture and Steadfastness Argument

To partially formalize the above conceptual argument, we will assume the formal agents model introduced earlier.

D.4.1 Toward a Formalization of GOLEM

We will use the notation $[A \to B]$ to denote the space of functions mapping space A to space B. Also, in cases where we denote a function signature via Φ_X, we will use X to denote the space of all programs embodying functions of that signature; e.g. GE is the space of all functions fulfilling the specification given for Φ_{GE}.

The GOLEM architecture may be formally defined as follows.

- The Historical Repository $\mathcal{H}_t$ is a subset of the history x_{0-t}
- An Operating Program is a program embodying a function $\Psi_{OP} : \mathcal{H} \to \mathcal{A}$. That is, based on a history (specifically, the one contained in the Historical Repository at a given point in time), it generates actions
- A Memory Manager is a program embodying a function so that $\Psi_{MM}(\mathcal{H}_t, x_t) = \mathcal{H}_{t+1}$
- A Goal Evaluator is a program embodying a function $\Psi_{GE} : \mathcal{H} \to [0, 1]$. That is, it maps histories (hypothetical future histories, in the GOLEM architecture) into real numbers representing utilities
- A Goal Evaluator Operating Program is an element of class GE
- A Searcher is a program embodying a function $\Psi_{SR} : [\mathcal{P} \to [0, 1]] \to \mathcal{P}$. That is, it maps "fitness functions" on program space into programs.
- A Predictor is a program embodying a function $\Psi_{PR} : OP \times GE \times \mathcal{H} \to [0, 1]$
- A Tester is a program embodying a function $\Psi_{TR} : PR \times \mathcal{H} \to [0, 1]$, where the output $[0, 1]$ is to be interpreted as the output of the prediction

- A Metaprogram is a program embodying a function $\Psi_{MP} : SR \times H \times PR \times TR \times GE^2 \times MM \to SR \times PR \times OP \times GE \times MM$. The GE in the output, and one of the GEs in the input, are GEOPs.

The operation of the Metaprogram is as outlined earlier; and the effectiveness of the architecture may be assessed as its average level of goal achievement as evaluated by the GE, according to some appropriate averaging measure.

As discussed above, a *fixed* GOLEM assumes a fixed GoalEvaluator, Tester and Metaprogram, and a fixed structure for the Historical Repository, and lets everything else adapt. One may also define an *adaptive* GOLEM variant in which *everything* is allowed to adapt, and this will be discussed below, but the conceptual steadfastness argument made above applies only to the fixed-architecture variant, and the formal proof below is similarly restricted.

Given the above formulation, it may be possible to prove a variety of theorems about GOLEM's steadfastness under various assumptions. We will not pursue this direction very far here, but will only make a few semi-formal conjectures, proposing some semi-formal propositions that we believe may result in theorems after more work.

D.4.2 Some Conjectures about GOLEM

The most straightforward cases in which to formally explore the GOLEM architecture are not particularly realistic ones. However, it may be worthwhile to begin with less realistic cases that are more analytically tractable, and then proceed with the more complicated and more realistic cases.

Conjecture D.1. Suppose that

- The Predictor is optimal (for instance an AIXI type system)
- Memory management is not an issue: there is enough memory for the system to store all its experiences with reasonable access time
- The GE is sufficiently efficient that no approximative GEOP is needed
- The HR contains all relevant information about the world, so that at any given time, the Predictor's best choices based on the HR are the same as the best choices it would make with complete visibility into the past of the universe

Then, there is some time T so that from T onwards, GOLEM will not get any worse at achieving the goals specified in the GE, unless it shuts itself off.

The basic idea of Conjecture D.1 is that, under the assumptions, GOLEM will replace its various components only if the Predictor predicts this is a good idea, and the Predictor is assumed optimal (and the GE is assumed accurate, and the Historical Repository is assumed to contain as much information as needed). The reason one needs to introduce a time $T > 0$ is that the initial

programs might be clever or lucky for reasons that aren't obvious from the HR.

If one wants to ensure that $T = 0$ one needs some additional conditions:

Conjecture D.2. In addition to the assumptions of Conjecture D.1, assume GOLEM's initial choices of internal programs are optimal based on the state of the world at that time. Then, GOLEM will never get any worse at achieving the goals specified in the GE, unless it shuts itself off.

Basically, what this says is: If GOLEM starts off with an ideal initial state, and it knows virtually everything about the universe that's relevant to its goals, and the Predictor is ideal – then it won't get any worse as new information comes in; it will stay ideal. This would be nice to know as it would be verification of the sensibleness of the architecture, but, isn't much practical use as these conditions are extremely far from being achievable.

Furthermore, it seems likely that

Conjecture D.3. Suppose that

- The Predictor is nearly optimal (for instance an $AIXI^{tl}$ type system)
- Memory management is not a huge issue: there is enough memory for the system to store a reasonable proportion of its experiences with reasonable access time
- The approximative GEOP is place is very close to accurate
- The HR contains a large percentage of the relevant information about the world, so that at any given time, the Predictor's best choices based on the HR are roughly same as the best choices it would make with complete visibility into the past of the universe

Then, there is some time T so that from T onwards, GOLEM is **very unlikely** to get **significantly** worse at achieving the goals specified in the GE, unless it shuts itself off.

Basically, this says that if the assumptions of Conjecture D.1 are weakened to approximations, then the conclusion also holds in an approximate form. This also would not be a practically useful result, as the assumptions are still too strong to be realistic.

What might we be able to say under more realistic assumptions? There may be results such as

Conjecture D.4. Assuming that the environment is given by a specific probability distribution μ, let

- δ_1 be the initial *expected* error of the Predictor assuming μ, and assuming the initial GOLEM configuration
- δ_2 be the initial *expected* deviation from optimality of the MM, assuming μ and the initial GOLEM configuration

- δ_3 be the initial *expected* error of the GEOP assuming μ and the initial GOLEM configuration
- δ_4 be the initial *expected* deviation from optimality of the HR assuming μ and the initial GOLEM configuration

Then, there are $\epsilon > 0$ and $p \in [0, 1]$ so that GOLEM has odds $< p$ of getting worse at achieving the goals specified in the GE by more than ϵ, unless it shuts itself off. The values ϵ and p may be estimated in terms of the δ values, using formulas that may perhaps be made either dependent on or independent of the environment distribution μ.

My suspicion is that, to get reasonably powerful results of the above form, some particular assumptions will need to be made about the environment distribution μ – which leads up to the interesting and very little explored problem of formally characterizing the probability distributions describing the "human everyday world."

D.5 Comparison to a Reinforcement Learning Based Formulation

Readers accustomed to reinforcement learning approaches to AI [SB98] may wonder why the complexity of the GOLEM meta-architecture is necessary. Instead of using a "goal based architecture" like this, why not just simplify things to

- Rewarder
- Operating Program
- Searcher
- Metaprogram

where the Rewarder issues a certain amount of reward at each point in time, and the Metaprogram: invokes the Searcher to search for a program that maximizes expected future reward, and then installs this program as the Operating Program (and contains some parameters balancing resource expenditure on the Searcher versus the Operating Program)?

One significant issue with this approach is that ensuring conservative meta-architecture preservation, based on reward signals, seems problematic. Put simply: in a pure RL approach, in order to learn that mucking with its own architecture is bad, the system would need to muck with its architecture and observe that it got a negative reinforcement signal. This seems needlessly dangerous! One can work around the problem by assuming an initial OP that has a bias toward conservative meta-architecture preservation. But then if one wants to be sure this bias is retained over time, things get complicated. For the system to learn via RL that removing this bias is bad, it would need to try it and observe that it got a negative reinforcement signal.

One could try to achieve the GOLEM within a classical RL framework by stretching the framework somewhat (RL++ ?) and

- allowing the Rewarder to see the OP, and packing the Predictor and GoalEvaluator into the Rewarder. In this case the Rewarder is tasked with giving the system a reward based on the satisfactoriness of the predicted outcome of running its Operating Program.
- allowing the Searcher to query the Rewarder with hypothetical actions in hypothetical scenarios (thus allowing the Rewarder to be used like the GoalEvaluator!)

This RL++ approach is basically the GOLEM in RL clothing. It requires a very smart Rewarder, since the Rewarder must carry out the job of predicting the probability of a given OP giving rise to a given world-state. The GOLEM puts all the intelligence in one place, which seems simpler. In RL++, one faces the problem of how to find a good Predictor, which may be solved by putting another Searcher and Metaprogram inside the Rewarder; but that complicates things inelegantly.

Note that the Predictor and GoalEvaluator are useful in RL++ specifically because we are assuming that in RL++ the Rewarder can see the OP. If the Rewarder can see the OP, it can reward the system for what it's *going to do* in the future if it keeps running the same OP, under various possible assumptions about the environment. In a strict RL design, the Rewarder cannot see the OP, and hence it can only reward the system for what it's going to do based on chancier guesswork. This guesswork might include guessing the OP from the system's actions – but note that, if the Rewarder has to *learn* a good model of what program the system is running via observing the system's actions, it's going to need to observe a *lot* of actions to get what it could get automatically by just seeing the OP. So the learning of the system can be much, much faster in many cases, if the Rewarder gets to see the OP and make use of that knowledge. The Predictor and GoalEvaluator are a way of making use of this knowledge.

Also, note that in GOLEM the Searcher can use the Rewarder to explore hypothetical scenarios. In a strict RL architecture this is not possible directly; it's possible only via the system in effect building an internal model of the Rewarder, and using it to explore hypothetical scenarios. The risk here is that the system builds a poor model of the Rewarder, and thus learns less efficiently.

In all, it seems that RL is not the most convenient framework for thinking about architecture-preserving AGI systems, and looking at "goal-oriented architectures" like GOLEM makes things significantly easier.

D.6 Specifying the Letter and Spirit of Goal Systems (Are Both Difficult Tasks)

Probably the largest practical issue arising with the GOLEM meta-architecture is that, given the nature of the real world, it's hard to estimate how well the Goal Evaluator will do its job! If one is willing to assume GOLEM, and if a proof corresponding to the informal argument given above can be found, then the"predictably beneficial" part of the problem of "creating predictably beneficial AGI" is largely pushed into the problem of the GoalEvaluator.

This makes one suspect that the *hardest* problem of making predictably beneficial AGI probably isn't "preservation of formally-defined goal content under self-modification." This may be hard if one enables total self-modification, but it seems it may not be *that* hard if one places some fairly limited restrictions on self-modification, as is done in GOLEM, and begins with an appropriate initial condition.

The *really* hard problem, it would seem, is how to create a GoalEvaluator that implements the desired goal content – and that updates this goal content as new information about the world is obtained, and as the world changes ... in a way that preserves the spirit of the original goals even if the details of the original goals need to change as the world is explored and better understood. Because the "spirit" of goal content is a very subtle and subjective thing.

The intelligent updating of the GEOP, including in the GOLEM design, will not update the original goals, but it will creatively and cleverly apply them to new situations as they arise – but it will do this according to Occam's Razor based on its own biases rather than necessarily according to human intuition, except insofar as human intuition is encoded in the base GEOP or the initial Searcher. So it seems sensible to expect that, as unforeseen situations are encountered, a GOLEM system will act according to learned GEOPs that are rationally considered "in the spirit of the base GEOP", but that may interpret that "spirit" in a different way than most humans would. These are subtle issues, and important ones; but in a sense they're "good problems to have", compared to problems like evil, indifferent or wireheaded [†] AGI systems.

D.7 A More Radically Self-Modifying GOLEM

It's also possible to modify the GOLEM design so as to enable it to modify the GEOP more radically – still with the intention of sticking to the spirit of the base GEOP, but allowing it to modify the "letter" of the base GEOP so as to preserve the "spirit." In effect this modification allows GOLEM to

[†] A term used to refer to situations where a system rewires its reward or goal-satisfaction mechanisms to directly enable its own maximal satisfaction

decide that it understands the essence of the base GEOP better than those who created the particulars of the base GEOP. This is certainly a riskier approach, but it seems worth exploring at least conceptually.

The basic idea here is that, where the base GEOP is uncertain about the utility of a world-state, the "inferred GEOP" created by the Searcher is allowed to be more definite. If the base GEOP comes up with a probability distribution P in response to a world-state W, then the inferred GEOP is allowed to come up with Q so long as Q is sensibly considered a refinement of P.

To see how one might formalize this, imagine P is based on an observation-set O_1 containing N observations. Given another distribution Q over utility values, one may then ask: What is the smallest number K so that one can form an observation set O_2 containing O_1 plus K more observations, so that Q emerges from O_2? For instance, if P is based on 100 observations, are there 10 more observations one could make so that from the total set of 110 observations, Q would be the consequence? Or would one need 200 more observations to get Q out of O_2?

Given an error $\epsilon > 0$, let the minimum number K of extra observations needed to create an O_2 yielding Q within error ϵ, be denoted $obs_\epsilon(P, Q)$. If we assume that the inferred GEOP outputs a confidence measure along with each of its output probabilities, we can then explore the relationship between these confidence values and the *obs* values.

Intuitively, if the inferred GEOP is *very* confident, this means it has a lot of evidence about Q, which means we can maybe accept a somewhat large $obs(P, Q)$. On the other hand, if the inferred GEOP is not very confident, then it doesn't have much evidence supporting Q, so we can't accept a very large $obs(P, Q)$.

The basic idea intended with a "confidence measure" here is that if $inferred\_geop(W)$ is based on very little information pertinent to W, then $inferred\_geop(W).confidence$ is small. The Tester could then be required to test the accuracy of the Searcher at finding inferred GEOPs with accurate confidence assessments: e.g. via repeatedly dividing the HistoricalRepository into training vs. test sets, and for each training set, using the test set to evaluating the accuracy of the confidence estimates produced by inferred GEOPs obtained from that training set.

What this seems to amount to is a reasonably elegant method of allowing the GEOP to evolve beyond the base GEOP in a way that is basically "in the spirit of the base GEOP." But with this kind of method, we're not necessarily going to achieve a long-term faithfulness to the base GEOP. It's going to be more of a "continuous, gradual, graceful transcendence" of the base GEOP, it would seem. There seems not to be any way to let the $inferred\_GEOP$ refine the $base\_GEOP$ without running some serious risk of the $inferred\_GEOP$ violating the "spirit" of the $base\_GEOP$. But what one gets in exchange for this risk is a GOLEM capable of having crisper goal evaluations, moving to-

ward lower-entropy utility distributions, in those cases where the base GEOP is highly uncertain.

That is, we can create a GOLEM that knows what it wants better than its creators did – but the cost is that one has to allow the system some leeway in revising the details of its creators' ideas based on the new evidence it's gathered, albeit in a way that respects the evidence its creators brought to bear in making the base GEOP.

D.8 Concluding Remarks

What we've sought to do here, in this speculative futuristic appendix, is to sketch a novel approach to the design of AGI systems that can massively improve their intelligence yet without losing track of their initial goals. While we have not proven rigorously that the GOLEM meta-architecture fulfills this specification, have given what seems to be a reasonable, careful informal argument, along with some semi-formal conjectures; and proofs along these lines will be pursued for later publications. T

It's clear that GOLEM can be wrapped around practical AGI architectures like CogPrime – and in that sense GOLEM is a natural extension of the remarks on self-modifying CogPrime systems from Chapter **??**. But the major open question is, how powerful do these architectures need to be in order to enable GOLEM to fulfill its potential as a meta-architecture for yielding significant ongoing intelligence improvement together with a high probability of goal system stability. The risk is that the rigors of passing muster with the Tester are sufficiently difficult that the base AGI architecture (CogPrime or whatever) simply doesn't pass muster, so that the base operating programs are never replaced, and one gets goal-system preservation without self-improvement. Neither our theory nor our practice is currently advanced enough to resolve this question, but it's certainly an important one. One approach to exploring these issues is to seek to derive a variant of CogPrime or some other practical AGI design as a specialization of GOLEM, rather than trying to study the combination of GOLEM with a separately defined AGI system serving as its subcomponent.

There is also the open worry of what happens when the system shuts down. Hypothetically, if a GOLEM system as described above were in a battle situation, enemies could exploit its propensity to shut down when its hardware is compromised. A GOLEM system with this property would apparently be at a disadvantage in such a battle, relative to a GOLEM system that avoided shutting down and instead made the best possible effort to repair its hardware, even if this wound up changing its goal system a bit. So, the particular safety mechanism used in GOLEM to prevent dangerous runaway self-improvement, would put a GOLEM at an evolutionary disadvantage. If a GOLEM system becomes intelligent before competing systems, and achieves

massively greater power and intelligence than any competing "startup" AGI system could expect to rapidly achieved, then this may be a nonissue. But such eventualities are difficult to foresee in detail, and devolve into generic futurist speculation.

Finally, the dichotomy between the fixed and adaptive GOLEM architectures highlights a major strategic and philosophical issue in the development of advanced AGI systems more broadly. The fixed GOLEM can grow far beyond humans in its intelligence and understanding and capability, yet in a sense, remains rooted in the human world, due to its retention of human goals. Whether this is a positive or negative aspect of the design is a profound nontechnical issue. From an evolutionary perspective, one could argue that adaptive GOLEMs will have greater ability to accumulate power due to their fewer limitations. However, a fixed GOLEM could hypothetically be created, with part of its goal system being to inhibit the creation of adaptive GOLEMs or other potentially threatening AGI systems. Here however we venture yet further into the territory of science fiction and speculative futurology, and we will leave further such discussion for elsewhere.

Appendix E
Lojban++: A Novel Linguistic Mechanism for Teaching AGI Systems

E.1 Introduction

Human "natural language" is unnatural to an AGI program like CogPrime . Yet, understanding of human language is obviously critical to any AGI system that wants to interact flexibly in the human world, and/or that wants to ingest the vast corpus of knowledge that humans have created and recorded. With this in mind, it is natural to explore humanly-unnatural ways of granting AGI systems knowledge of human language; and we have done much of this in the previous appendices, discussing the use of linguistic resources that are clearly different in nature from the human brain's in-built linguistic biases. In this appendix we consider yet another humanly-unnatural means of providing AGI systems with linguistic knowledge: the use of the constructed language Lojban (or, more specifically, its variant Lojban++), which occupies an interesting middle ground between formal languages like logic and programming languages, and human natural languages. We will argue that communicating with AGI systems in Lojban++ may provide a way of

- providing AGI systems with experientially-relevant commonsense knowledge, much more easily than via explicitly encoding this knowledge in logic
- teaching AGI systems natural language much more quickly than would otherwise be possible, via communicating with AGIs in parallel using natural language and Lojban++

To put it more precisely: the essential goal of Lojban++ is to constitute a language for efficient, minimally ambiguous, and user-friendly communications between humans and suitably-constructed AI software agents such as CogPrime s. Another way to think about the Lojban++ approach is that it allows an AGI learning/teaching process that dissociates, to a certain extent, "learning to communicate with humans" from "learning to deal with the peculiarities of human languages." Similar to Lojban on which it is based,

Lojban++ may also be used for communication between humans, but this interesting possibility will not be our focus here.

Some details on the particulars of the Lojban++ language proposal, aimed at readers familiar with Lojban, are given at the end of this appendix. In the initial portions of the appendix we describe Lojban++ and related ideas at a more abstract level, in a manner comprehensible to readers without prior Lojban background.

E.2 Lojban versus Lojban++

Lojban is itself an outgrowth of another constructed language, Loglan, created by Dr. James Cooke Brown around 1955 and first widely announced in a 1960 Scientific American article [Bro60]. Loglan is still under development but now is not used nearly as widely as Lojban. First separated from Loglan in 1987, Lojban is a constructed language that lives at the border between natural language and computing language. It is a "natural-like language" in that it is speakable and writeable by humans and may be used by humans to discuss the same range of topics as natural languages. Lojban has a precise, specified formal syntax that can be parsed in the same manner as a programming language, and it has a semantics, based on predicate logic, in which ambiguity is carefully controlled. Lojban semantics is not completely unambiguous, but it is far less ambiguous than that of any natural language, and the careful speaker can reduce ambiguity of communication almost to zero with far less effort than in any natural language. On the other hand, Lojban also permits the speaker to utilize greater ambiguity when this is desirable in order to allow compactness of communication.

Many individuals attempting to learn and use Lojban have found, however, that it has two limitations. The Lojban vocabulary is unfamiliar and difficult to learn - though no more so than that of any other language belonging to a language family unfamiliar to the language learner. And, more seriously, the body of existing Lojban vocabulary is limited compared to that of natural languages, making Lojban communication sometimes slow and difficult. When using Lojban, one must sometimes pause to concoct new words (according to the Lojban principles of word construction), which can be fun, but is much like needing to stop over and over to build new tools in the context of using one's toolkit to build something; and is clearly not optimal from the perspective of teaching AGI systems.

To address these issues, Lojban++ constitutes a combination of Lojban syntax and Lojban vocabulary, extended with English vocabulary. So in a very rough sense, it may perhaps be understood as a pidgin of Lojban and English. Lojban++ is less elegant than Lojban but significantly easier to learn, and much easier to use in domains to which Lojban vocabulary has not yet been extended. In short, the goal of Lojban++ is to combine the mathe-

matical precision and pragmatic ontology that characterize Lojban, with the usability of a natural language like English with its extensive vocabulary.

An extensive formal treatment of Lojban grammar has been published [Cow97], and while there is no published hard-copy Lojban dictionary, there is a website jbovlaste.lojban.org/ that serves this purpose and which is frequently updated as new coinages are created and approved by the Logical Language Group, a standing body charged with the maintenance of the language.

Although Lojban has not been adopted nearly as widely as Esperanto (an invented language with several hundred thousand speakers), the fact that there is a community of several hundred speakers, including several dozen who are highly fluent at least in written Lojban, is important. The decades of communicative practice that have occurred within the Lojban community have been invaluable for refining the language. This kind of practice buys a level of maturity that cannot be obtained in a shorter period of time via formal analysis or creative invention. For example, the current Lojban treatment of quantifiers is arguably vastly superior to that of any natural language [Cow97], but that was not true in 1987 when it excelled more in mathematical precision than in practical usability. The current approach evolved through a series of principled revisions suggested from experience with practical conversation in Lojban. Any new natural-like language that was created for human-CogPrime or CogPrime -CogPrime communication would need to go through a similar process of iterative refinement through practical use to achieve a similar level of usability.

E.3 Some Simple Examples

Now we give some examples of Lojban++. While these may be somewhat opaque to the reader without Lojban experience, we present them anyway just to give a flavor of what Lojban++ looks like; it would seem wrong to leave the discussion purely abstract.

Consider the English sentence,

When are you going to the mountain?

When written in Lojban, it looks like:

do cu'e klama le cmana

In Lojban++, with the judicious importation of English vocabulary, it takes a form more recognizable to an English speaker:

you cu'e go le cmana

A fairly standard predicate logic rendition of this, derived by simple, deterministic rules from the Lojban++ version, would be

```
atTime(go(you, mountain), ?X)
```

Next, consider the more complex English sentence,

When are you going to the small obsidian mountain?

In Lojban, there is no word for obsidian, so one needs to be invented (perhaps by compounding the Lojban words for "glass" and "rock," for example), or else a specific linguistic mechanism for quoting non-Lojban words needs to be invoked. But in Lojban++ one could simply say,

you cu'e go le small obsidian mountain

The construct "small obsidian mountain" is what is called a Lojban tanru, meaning a compound of words without a precisely defined semantics (though there are recognized constraints on tanru semantics based on the semantics of the components [**?**]). Alternatively, using the Lojban word, marji, which incorporates explicit place structure (x1= material/stuff/matter of composition x2), a much less ambiguous translation is achieved:

you cu'e go le small mountain poi marji loi obsidian

in which "poi marji loi obsidian" means "that is composed of [a mass of] obsidian." This illustrates the flexible ambiguity achievable in Lojban. One can use the language in a way that minimizes ambiguity, or one can selectively introduce ambiguity in the manner of natural languages, when desirable.

The differences between Lojban and Lojban++ are subtler than it might appear at first. It is key to understand hat Lojban++ is not simply a version of Lojban with English character-sequences substituted for Lojban character-sequences. A critical difference lies in the rigid, pre-determined argument structures associated with Lojban words. For instance, the Lojban phrase

klama fi la .atlantas. fe la bastn. fu le karce

corresponds to the English phrase

that which goes from Atlanta to Boston by car

To say this in Lojban++ without using "klama" would require

go fi'o source Atlanta fi'o destination Boston fi'o vehicle car

which is much more awkward. On the other hand, one could also avoid the awkward Lojban treatment of English proper nouns and say

klama fi la Atlanta fe la Boston fu le car

or

klama fi la Atlanta fe la Boston fu le karce

It's somewhat a matter of taste, but according to ours, the latter most optimally balances simplicity with familiarity. The point is that the Lojban word "klama" comes with the convention that its second argument (indexed by "fi") refers to the source of the going, its third argument (indexed by "fe") refers to the destination of the going, and its fifth argument (indexed by "fu") refers to the method of conveyance. No such standard argument-structure template exists in English for "go", and hence using "go" in place of "klama" requires the use of the "fi'o" construct to indicate the slot into which each of the arguments of "go" is supposed to fall.

The following table gives additional examples, both in Lojban and Lojban++.

English	I eat the salad with croutons
Lojban	mi citka le salta poi mixre lo sudnabybli
Lojban++	mi eat le salad poi mixre lo crouton mi eat le salad poi contain lo crouton
English	I eat the salad with a fork
Lojban	mi citka le salta sepi'o lo forca
Lojban++	mi eat le salad sepi'o lo fork
English	I will drive along the road with the big trees
Lojban	mi litru le dargu poi lamji lo barda tricu
Lojban++	mi ba travel fi'o vehicle lo car fi'o route le road poi adjacent lo so'i big tree mi ba litru fi lo car fe le road poi adjacent lo so'i big tree mi ba drive fi'o route le road poi adjacent lo so'i big tree
English	I will drive along the road with great care
Lojban	mi litru le dargu ta'i lo nu mi mutce kurji
Lojban++	mi ba drive fi'o route le road ta'i lo nu mi much careful mi ba litru le road ta'i lo nu mi much careful
English	I will drive along the road with my infrared sensors on
Lojban	mi ba litru le dargu lo karce gi'e pilno le miktrebo'a terzga
Lojban++	mi litru le road lo car gi'e use le infrared sensor mi litru le road lo car gi'e pilno le infrared te zgana mi drive fi'o vehicle lo car fi'o route le road gi'e use le infrared sensor
English	I will drive along the road with the other cars
Lojban	mi litru le dargu fi'o kansa lo drata karce
Lojban++	mi ba drive fi'o route le road fi'o kansa lo so'i drata car mi ba drive fi'o route le road fi'o with lo so'i drata car mi ba litru le road fi'o kansa lo so'i drata car

E.4 The Need for Lojban Software

In order that Lojban++ be useful for human-CogPrime communications, parsing and semantic mapping software need to be produced for the language, building on existing Lojban software.

There is a fully functional Lojban parser based on a parsing expression grammar (Powell, no date specified), as well as an earlier parser based on BNF grammar. (And, parenthetically, the observation that Lojban is more conveniently formulated in PEG (Parsing Expression Grammar) form is in itself a nontrivial theoretical insight.) The creation of a Lojban++ parser based on the existing Lojban parser, is a necessary and a relatively straightforward though not trivial task.

On the other hand, no software has yet been written for formal semantic interpretation ("semantic mapping") of Lojban expressions - which is mainly because Lojban has primarily been developed as an experimental language for communication between humans rather than as a language for human-CogPrime communication. Such semantic mapping software is necessary to complete the loop between humans and AI reasoning programs, enabling powerful cognitive and pragmatic interplay between humans and CogPrime s. For Lojban++ to be useful for human-CogPrime interaction, this software must be created and must go in both directions: from Lojban++ to predicate logic and back again. As Lojban++ is a superset of Lojban, creating such software for Lojban++ will automatically include the creation of such software for Lojban proper.

There promises to be some subtlety in this process, but not on the level that's required to semantically map human language. What is required to connect a Lojban++ parser with the RelEx NLP framework as described in Chapter 44 is essentially a mapping between

- the Lojban cmavo (structure word) and the argument-structure of lojban gismu (root word)
- FrameNet frame-elements, and a handful of other CogPrime relationships (e.g. for dealing with space, time and inheritance)

These mappings must be built by hand, which should be time-consuming, but on the order of man-weeks rather than man-years of effort.* Once this is done, then Lojban++ can be entered into CogPrime essentially *as English would be if the RelEx framework worked perfectly.* The difficulties of human language processing will be bypassed, though still – of course – leaving the difficulties of commonsense reasoning and contextual interpretation.

For example, the Lojban root word *klama* is defined as

x1 comes/goes to destination x2 from origin x3 via route x4 using means/vehicle x5.

* Carrying out the following mapping took a few minutes, so carrying out similar mappings for 800 FrameNet frames should take no more than a couple weeks of effort.

This corresponds closely to the FrameNet frame Motion, which has elements

- Theme (corresponding to x1 in the above Lojban definition)
- Source (x2)
- Goal (x3)
- Path (x4)
- Carrier (x5)

The Motion FrameNet frame also has some elements that klama lacks, e.g. Distance and Direction, which could of course be specified in Lojban using explicit labeling.

E.5 Lojban and Inference

Both Lojban and Lojban++ can be straightforwardly translated into predicate logic format (though the translation is less trivial in the case of Lojban++, as a little bit of English-word disambiguation must be done). This means that as soon as Lojban++ semantic mapping software is constructed, it will almost immediately be possible for CogPrime systems to reason about knowledge communicated to them in Lojban. This aspect of Lojban has already been explored in a preliminary way by Speer and Havasi's [**?**] JIMPE software application, which involves a semantic network guiding logical reasoning, Lojban parsing and Lojban language production. While JIMPE is a relatively simplistic prototype application, it is clear that more complex example of Lojban-based artificial inference are also relatively straightforwardly achievable via a conceptually similar methodology.

An important point to consider in this regard is that Lojban/Lojban++ contains two distinct aspects:

1. an ontology of predicates useful for representing commonsense knowledge (represented by the Lojban cmavo along with the most common Lojban content words)
2. a strategy for linearizing nested predicates constructed using these cmavo into human-pronounceable and -readable strings of letters or phonemes.

The second aspect is of no particular value for inference, but the first aspect is. We suggest that the Lojban++ ontology provides a useful framework for knowledge representation that may be incorporated at a fundamental level into any AI system that centrally utilizes predicate logic or a similar representation. While overlapping substantially with FrameNet, it has a level of commonsensical completeness that FrameNet does not, because it has been refined via practice to be useful for real-world communication. Similarly, although it is smaller than Cyc, it is more judiciously crafted. Cyc contains a lot of knowledge not useful for everyday communication, yet has various

lacunae regarding the description of everyday objects and events – because no community has ever seriously tried to use it for everyday communication.

E.5.1 Lojban versus Predicate Logic

In the context of Lojban++ and inference, it is interesting to compare Lojban++ formulations with corresponding predicate logic formulations. For example, consider the English sentence

Hey, I just saw a bunch of troops going into the woods. What do you want me to do?

translates into the Lojban

ju'i do'u mi pu zi viska lo nu so'i lo sonci cu nenkla le ricfoi .i do djica lo nu mi mo

or the Lojban++

Hey do'u mi pu zi see lo nu so'i lo soldier cu enter le forest .i do want lo nu mi mo

which literally transcribed into English would be something like

Hey! [vocative terminator] I [past] [short time] see an event of (many soldiers enter forest). You want event (me what?)

Omitting the "hey," a simple and accurate predicate logic rendition of this sentence would be

$$past(\$X) \wedge short\_time(\$X) \wedge (\$X = see(me, \$Y)) \wedge (\$Y = event(enter(\$Z, forest))) \wedge soldier(\$Z) \wedge many(\$Z) \wedge want(you, event(?W(me))$$

where ?W refers to a variable being posed as a question be answered, and X and so forth refer to internal variables. The Lojban and Lojban++ versions have the same semantics as the predicate logic version, but are much simpler to speak, hear and understand due to the lack of explicit variables.

E.6 Conclusion

Hopefully the above exposition of Lojban++, though incomplete, was sufficient to convince you that teaching "infant-level" or "child-level" AGIs about the world using Lojban++ would be significantly easier than teaching doing

so using English or other natural languages. The question then is whether this difference makes any difference.

One could counter-argue that, if an AGI were smart enough to really learn to interpret Lojban++, then it would be smart enough to learn to interpret English as well, with only minor additional effort. In sympathy with this counter-argument is the fact that successfully mapping Lojban++ utterances into predicate logic expressions, and representing these predicate logic expressions in an AI's knowledge base, does not in itself constitute any serious "understanding" of the Lojban++ utterances on the part of the AI system.

However, this counter-argument ignores the "chicken and egg problem" of common-sense knowledge and language understanding. If an AGI understands natural language then it can be taught human common-sense knowledge via direct linguistic instruction. On the other hand, it is also clear that a decent amount of common-sense knowledge is a prerequisite for adequate natural language understanding (for such tasks as parse selection, semantic disambiguation and reference resolution, for example). One response to this is to appeal to feedback, and argue that common-sense knowledge and linguistic understanding are built to arise and grow together. We believe this is largely true, and yet that there may also be additional dynamics at play in the developing human mind that accelerate the process, such as inbuilt inductive biases regarding syntax. In an AGI context, one way to accelerate the process may be to use Lojban++ to teach the young AGI system commonsense knowledge, which then may help it to more easily penetrate the complexities and ambiguities of natural language.

This assumes, of course, that the knowledge gained by the system from being instructed in Lojban++ is genuine knowledge rather than merely empty, ungrounded tokens. For this reason, we suggest, one viable learning project may be to teach an AGI system using Lojban++ in the context of shared embodied experience in a real or simulated environment. This way Lojban++ expressions may be experientially grounded and richly understood, potentially allowing commonsense knowledge to form in an AGI's knowledge base, in a way that can be generalized and utilized to aid with various learning tasks including learning natural language.

Another interesting teaching strategy may be to present an AGI system with semantically roughly equivalent English and Lojban sentences, especially ones that are pertinent to the system's experiences. Since the system can interpret the Lojban sentences with minimal ambiguity (especially by using the experienced context to reduce any ambiguities remaining after parsing, due to tanru), it will then know the correct interpretation of the English sentences, which will provide it with very helpful "training data" that it can then generalize from to help it understand other English sentences.

E.7 Postscript: Basic Principles for Using English Words in Lojban++

This section reviews the key principles by which Lojban++ incorporates English words into Lojban, and discusses some other small additions that Lojban++ makes to Lojban. It is intended mainly for readers who are familiar with Lojban.

A note is perhaps appropriate here regarding the right approach to learning Lojban++ at present. Lojban++ is a variant of Lojban, and no systematic teaching materials for Lojban++ yet exist. Therefore, at the moment the only way to learn Lojban++ is to learn the basics of Lojban, and then learn the differences (note however, that the "basics of Lojban" as meant here does not necessarily include a broad mastery of Lojban vocabulary beyond the cmavo or "structure words"). Assuming Lojban++ is used for teaching AGI systems as proposed here, relevant teaching materials should also be developed. There is no need to write a book-length grammar for Lojban++ comparable to [**?**], however, since the prinicples of Lojban++ grammar are all drawn from Lojban.

Finally, a necessary caveat: Lojban++ is not yet refined through practice, so it should be assumed that the specifics described in this Appendix are likely to be subjected to change through experience, as the language is used and developed.

This list of principles will likely be extended and refined through usage.

1. Content words only! English words that are about syntactic relationship have no place in Lojban++.
2. No "being verbs" or "helping verbs." The English "is" and its conjugations have no place in Lojban++, for example.
3. All Lojban++ cares about is the main part of an English word. None of the English markers for tense, person or number should be used, when importing an English word into Lojban++. For instance, English verbs used must be in the infinitival form; English nouns must be used in the singular form. For instance, "run" not "runs" or "ran"; "pig" not "pigs."
4. English adverbs are not used except in rare cases where there is no adjectival form; where there is an adjectival form it is used instead – e.g. "scary" not "scarily."

To lapse into Lojban lingo, English words must be used in Lojban++ as brivla. Tense, number and so forth are supposed to be added onto these brivla using the appropriate Lojban cmavo. The Lojban++ parser will assume that any non-Lojban word encountered, if not specifically flagged as a proper name (by the cmavo "la"), is an English word intended to be interpreted as a brivla. It will not do any parsing of the word to try to interpret tense, number, adverbiality, etc.

Next, English idiomatic collocations, if used in written Lojban++, should be used with an underscore between the component words. For example:

New_York, run_wild, big_shot, etc. Without the underscore, the Lojban++ parser will assume that it is seeing a tanru (so that e.g. "big shot" is a type of "shot" that is modified by "big"). In spoken Lojban, the formally correct thing is to use the new cmavo "quay" to be discussed below; but in practice when using Lojban++ for human-human communication this may often be omitted.

Finally, a less formal guideline concerns the use of highly ambiguous English words, the use of obscure senses of English words, and the use of English words in metaphorical senses. All of these should be avoided. They won't confuse the Lojban++ parsing process, but they will confuse the Lojban++ semantic mapping process. If a usage seems like it would confuse an AI program without much human cultural experience, then try to avoid it. Don't say

you paint ti

to mean "paint" in the sense of "portray vividly", when you could say

you cu vivid bo describe ti

The latter will tell an AI exactly what's happening; the former may leave the AI wondering whether what's being depicted is an instance of description, or an instance of painting with an actual paintbrush and oils. Similarly, to say

you kill me

when you mean

you much amuse me

is not in the Lojban++ spirit. Yes, an AI may be able to figure this out by reference to dictionaries combined with contextual knowledge and inference, but the point of Lojban++ is to make communication simple and transparent so as to reduce the possibility for communication error.

E.8 Syntax-based Argument Structure Conventions for English Words

Next, one of the subtler points of Lojban++ involves the automatic assignment of Lojban argument-structures to English words. This is done via the following rules:

1. Nouns are interpreted to have one argument, which is interpreted as a member of the category denoted by the noun
    a. *la Ben human*

1. Adjectives/adverbs are taken to have two arguments: the first is the entity modified by the adjective/adverb, the second is the extent to which the modification holds

    a. *la Ben fat le slight*

1. Intransitive verbs are interpreted to have at least one argument, which is interpreted as the argument of the predicate represented by the verb

    a. *le cockroach die*

1. Transitive verbs are interpreted to have at least two arguments, the subject and then the object

    a. *la Ben kill le cockroach*

1. Ditransitive verbs are interpreted to have three arguments, and conventions must be made for each of these cases, e.g.

1. a. give x y z may be interpreted as "x give y to z"
        i. *la Ben give le death le cockroach*
    b. take x y z may be interpreted as "x takes y from z"
        i. *la Ben take le life le cockroach*

A rule of thumb here is that the agent comes first, the recipient comes last, and the object comes inbetween.

E.9 Semantics-based Argument Structure Conventions for English Words

The above syntax-based argument-structure conventions are valuable, but not sufficiently thorough to allow for fluent Lojban++ usage. For this reason a collection of semantics-based argument-structure conventions have been created, based mostly on porting argument-structures from related Lojban words to English vocabulary. The following list is the current working version, and is likely to be extended a bit during actual usage.

1. Plant or animal (moss, cow, pig)

    a. x1 is a W of species x2

2. Spatial relation (beneath, above, right, left)

    a. x1 is in relation W to x2, in reference frame x3

3. Dimension-dependent spatial descriptor (narrow, deep, wide, etc.)

    a. x1 is W in dimension x2, relative to standard x3

4. Unit (foot, hour, meter, mile)
    a. x1 is x2 W's by standard x3
5. Kinship or other interpersonal relationship (mother, father, uncle, boss)
    a. x1 is the W of x2
6. Thought-action (remember, think, intuit, know)
    a. x1 W's x2
    b. x1 W's x2 about x3
7. Creative product (poem, painting, book)
    a. x1 is a W about plot/theme/subject/pattern x2 by author x3 for intended audience x4
8. Physical action undertaken by one agent on another (touch, kick, kiss)
    a. x1 (agent) W's x2 with x3 [a locus on x1 or an instrument] at x4 [a locus on x2]
9. W denotes a type of substance, e.g. mush, paste, slime
    a. x1 is a W composed of x2
10. Instance of communication (ask, tell, command)
    a. x1 W's x2 with information content x3
11. Type of utterance (comment, question)
    a. x1 (text) is a W about subject x2 expressed by x3 to audience x4
12. Type of movement (walking, leaping, jumping, climbing)
    a. x1 (agent/object) W's to x2 from x3 in direction x4
13. Route, path, road, trail, etc.
    a. x1 is a W to x2 from x3 via/defined by points including x4 (set)
14. Nationality, culture etc.
    a. x1 reflects W in aspect x2
15. Type of event involving humans or other social agents (celebration, meeting, funeral)
    a. x1 partakes, with purpose x2, in event x3 of type W
16. Posture or mode of physical activity of an embodied agent (stand, sit, lie, stoop)

   a. x1 W's on surface x2

17. Type of mental construct (idea, thought, dream, conjecture, etc.)

   a. x1 is a W about x2 by mind x3

18. Type of event done by someone, potentially to someone else (accident, disaster, injury)

   a. x1 is a W done by x2 to x3

19. Comparative amount (half, third, double, triple)

   a. x1 is W of x2 in quality x3

20. Relation between an agent and a statement (assert, doubt, refute, etc.)

   a. x1 W's x2

21. Spatial relationship (far, near, close)

   a. x1 is W from x2 in dimension x3

22. Human emotion (happy, sad, etc.)

   a. x1 is W about x2

23. A physically distinct part of some physical object, including a body part

   a. x1 is a W on x2

24. Type of physical transformation (e.g. mash, pulverize, etc.)

   a. x1 [force] W's x2 into mass x3

25. Way of transmitting an object (push, throw, toss, fling)

   a. x1 W's object x2 to/at/in direction x3

26. Relative size indicator (big, small, huge)

   a. x1 is W relative to x2 by standard x3

E.10 Lojban gismu of clear use within Lojban++

There are some Lojban gismu (content words) which are clearly much more useful within Lojban++ than their English counterparts. Mostly this is because their argument structures involve more than two arguments, but occasionally it is because they involve a two-argument structure that happens

not to be well-captured by any English word (but is usually represented in English by a more complex construct involving one or more prepositions).

A list of roughly 300 gismu currently judged to be "essential" in this sense is at http://www.goertzel.org/papers/gismu_essential.txt, and a list of <50 additional gismu judged potentially very useful but not quite so essential is at urlhttp://www.goertzel.org/papers/gismu_useful.txt

E.11 Special Lojban++ cmavo

Next, there are some special cmavo (structure words) that are useful in Lojban++ but not present in ordinary Lojban. A few more Lojban++ cmavo may be added as a result of practical experience communicating using Lojban++; but these are it, for now.

E.11.1 qui

Pronounced "kwee", this is a cmavo used in Loglish to create words with unambiguous senses, as in the example:

pig qui animal

pig qui cop

The second English word in the compound is a sense-specifier. Generally this should only be used where the word-sense intended is not the one that would be most obviously expected given the context.

In some rare cases one might want two modifiers, using the form

(English word) qui (English word) qui (English word)

E.11.2 it , quu

The basic idea is that there is one special referential word in Lojban++ – "it" – which goes along with a reference-target-indicator "quu" (pronounced "kwuhh") which gives a qualitative indication of the referent of a given instance of "it," intended to narrow down the scope of the reference resolution process.

For instance, you could say

la Dr. Benjamin Goertzel cu proceed le playground. It quu man cu kill le dog. It cu eat le cat.

In this case, "it" is defined to refer to "Dr. Benjamin Goertzel", not to "man" generically. The "man" qualifier following the "quu" is intended to merely guide the listener's mind toward the right antecedent for the pronoun. It's not intended to explicitly define the pronoun. So, basically

it quu male

is the rough equivalent of the English "he", and

it quu female

is the rough equivalent of the English "she"

him/her/they

Finally, for sake of usability, it is worthwhile within Lojban++ to introduce the following shorthands

- him → it quu male
- her → it quu female
- ver → it quu person
- they → it quu people

(Note that "him" in Lojban++ thus plays the role of both "him" and "he" in English.)

E.11.3 quay

Pronounced "kway," this cmavo separates parts of an English collocation in speech, e.g.

big quay shot

It may often be omitted in informal speech; and in writing may be replaced by an underscore (big_shot).

Appendix F
PLN and the Brain

Co-authored with Cassio Pennachin

F.1 How Might Probabilistic Logic Networks Emerge from Neural Structures and Dynamics?

In this appendix, we digress briefly to explore how PLN constructs like inheritance and similarity relationships might emerge from brainlike structures like cell assemblies and neural activation patterns. This is interesting as speculative neuroscience, and also potentially valuable in the context of hybrid architectures, in terms of tuning the interrelationship between CogPrime 's Atomspace and neural net like systems such as DeSTIN. If nothing else, the ideas of this section serve as a conceptual argument why it makes sense to interface PLN representations and dynamics with CSDLN representations and dynamics. While conventionally formalized and discussed using different languages, these different approaches to knowledge and learning are actually not so far off as is commonly believed.

We restrict ourselves here to FOPLN, which does not involve explicit variables or quantifiers, and may be described as the logic of uncertain inheritance relationships. As in PLN higher-order logic reduces to first-order logic, this is actually all we need to deal with. A neural implementation of higher-order PLN follows from a neural representation of FOPLN plus a neural representation of higher-order functions such as the one suggested in Chapter 13 of Part 1.

As described above, the semantics of the term logic relationship "A inherits from B" or $A \rightarrow B$, is that when B is present, A is also present. The truth value of the relationship measures the percentage of the times that B is present, that A is also present. "A is similar to B" or $A \leftrightarrow B$, is a symmetrical version, whose truth value measures the percentage of the times that either one is present, that both are present. These are the relations we will deal with here.

How can this be tied in with the brain? Suppose we have two assemblies A1 and A2, and these are activated in the brain when the organism is presented

with stimuli in category C1 and C2 respectively (to take the simplest case of concepts, i.e. perceptual categories). Then, we may say that there is a *neural inheritance* $A1 \to A2$, whose probabilistic strength is the number w so that

$$P(A1's \text{ mean activation} > T \text{ at time } t) * w$$

best approximates

$$P(A2's \text{ mean activation} > T \text{ at time } t + \epsilon)$$

for an appropriate choice of ϵ and T. This weight w, intuitively, represents the conditional probability $P(A2 \text{ is active} | A1 \text{ is active})$.

In a similar way, if we have two assembly activation patterns P1 and P2, which are defined as specific types of activity patterns occurring in A1 and A2 respectively, and which correspond to categories C1 and C2 respectively, we can define a neural inheritance $P1 \to P2$, whose probabilistic truth value is the number w so that $P(P1 \text{ is present in } A1 \text{ at time } t) * w$ best approximates $P(P2 \text{ is present in } A2 \text{ at time } t + \epsilon)$, on average over various times t, and assuming a threshold T used to determine when a pattern is present in an assembly.

It is immediate that, if there is a virtual synapse between A1 and A2 or P1 and P2, there will also be a neural inheritance there. Furthermore there will be a monotone increasing algebraic relationship between the weight of the virtual synapse and the probability attached to the neural inheritance. Inhibitory virtual synapses will correspond to very low link probabilities; excitatory virtual synapses will correspond to high link probabilities.

However, we can have neural inheritance without any virtual synapse. This is a key point, as it lets us distinguish neural inheritance relationships that are *explicit* (that correspond to virtual synapses) from those that are *implicit* (that do not). And this leads us directly to probabilistic reasoning, which is about transforming implicit inheritance relationships into explicit ones. The fundamental inference rules of term logic, as described above, create new inheritance links from old ones. The conclusions are implicit in the premises but until the inference is done, they may not be explicitly contained in the knowledge base of the system doing the reasoning. Probabilistic reasoning in the brain, we suggest, is all about translating implicit neural inheritance relationships into explicit ones.

In first-order PLN the basic forms of inference are: revision (which in its simplest form is weighted-averaging), deduction, and inversion (which reverses the direction of a link, and in probabilistic term logic is essentially Bayes rule). Let us elaborate what these mean in terms of cell assemblies.

Suppose that A1 and A2 are two assemblies, and there is a neural inheritance between A1 and A2. Then, there will also be a neural inheritance between A2 and A1, with a truth value given by Bayes rule. And according to Hebbian learning if there is a virtual synapse $A1 \to A2$, there is likely to grow a virtual synapse $A2 \to A1$. And according to the approximate correla-

tion between virtual synapse weight and neural inheritance probability, this new virtual synapse from $A2 \to A1$ will have a weight corresponding to a probability approximating the one corresponding to the weight of $A1 \to A2$.

Similarly, suppose that A1, A2 and A3 are three assemblies. Then, if we have virtual synapses between $A1 \to A2$ and $A2 \to A3$, Hebbian learning suggests that a virtual synapse will grown between $A1 \to A3$. And what will the probability of the neural inheritance corresponding to this virtual synapse be? On average, it will be the probability one obtains by assuming the probabilities associated with $A1 \to A2$ and $A2 \to A3$ are independent. But, this means that on average, the probability associated with $A1 \to A3$ will accord with the value produced by the PLN deduction formula, which embodies precisely this independence assumption. Here, we have an additional source of error beyond what exists in the Bayes rule case; but, in the mean, the desired correspondence does hold.

So, according to the above arguments – which admittedly have been intuitive rather than mathematically rigorous –it would seem that we can build term logic inference between concepts out of Hebbian learning between neurons, if we assume cell assembly based knowledge representation, via introducing the conceptual machinery of virtual synapses and neural inheritance.

F.2 Avoiding Issues with Circular Inference

When one develops the ideas from the previous section, connecting uncertain term logic inference with neurodynamics, in more detail, only one possible snag arises. Existing computational frameworks for uncertain term logic inference utilize special mechanisms for controlling circular inference, and these mechanisms have no plausible neurological analogues. In this section we explore this issue and argue that it's not necessarily a big deal. In essence, our argument is that these biologically unnatural circularity-avoidance mechanisms are unnecessary in a probabilistic term logic system whose operations are guided by appropriate adaptive attention-allocation mechanisms. It's only when operating probabilistic term logic inference in isolation, in a manner that's unnatural for a resource-constrained intelligent system, that these circular-inference issues become severe.

We note however that this conclusion seems to be specific to probabilistic term logic, and doesn't seem to hold for NARS term logic, in which the circular inference problem may be more severe, and may in fact require a trail mechanism more strongly. We have not investigated this issue carefully.

To understand the circular inference problem, look at the triangles in Figure 34.1. It's easy to see that by performing deduction, induction and abduction in sequence, we can go around and around an inference triangle forever, combining the links in different orders, inferring each link in the triangle from the two others in different orders over and over again. What often happens

when you do this in a computer program performing uncertain term logic inference, however, is that after long enough the inference errors compound, and the truth values descend into nonsense. The solution taken in the NARS and PLN uncertain term logic inference engines is something called *inference trails*. Basically, each inheritance link maintains a trail, which is a list of the nodes and links used as premises in inferences determining its truth value. And a rule is put in place that the link L should not be used to adjust the truth value of the link M if M is in L's trail.

Trails work fine for computer programs implementing uncertain term logic, though managing them properly does involve various complexities. But, from the point of view of the brain, trails seem quite unacceptable. It would seem implausible to hypothesize that the brain somehow stores a trail along with each virtual synapse. The brain must have some other method of avoiding circular inferences leading to truth value noisification.

In order to explore these issues, we have run a number of experiments with trail-free probabilistic inference. The first of these involved doing inferences on millions of nodes and links (with nodes representing words and links derived via word co-occurrence probabilities across a text corpus). What we found was that, in practice, the severity of the circular-inference problem depended on the inference control strategy. When one implemented a strategy in which the amount of attention devoted to inference about a link L was proportional to an estimate of the amount of information recently gained by doing inference about L, then one did not run into particularly bad problems with circular inference. On the other hand, if one operated with a small number of nodes and links and repeatedly ran the same inferences over and over again on them, one did sometimes run into problems with truth value degeneration, in which the term logic formulas would cause link strengths to spuriously converge to 1 or 0.

To better understand the nature of these phenomena, we ran computer simulations of small Atomspaces involving nodes and Inheritance relations, according to the following idea:

1. Each node is assumed to denote a certain perceptual category
2. For simplicity, we assume an environment in which the probability distribution of co-occurrences between items in the different categories is stationary over the time period of the inference under study
3. We assume the collection of nodes and links has its probabilistic strengths updated periodically, according to some "inference" process
4. We assume that the results of the inference process in Step 3 and the results of incorporating new data from the environment (Step 2) are merged together ongoingly via a weighted-averaging belief-revision process

In our simulations Step 3 was carried out via executions of PLN deduction and inversion inference rules. The results of these simulations were encouraging: most of the time, the strengths of the nodes and links, after a while,

settled into a "fixed point" configuration not too distant from the actual probabilistic relationships implicit in the initial data. The final configuration was rarely equivalent to the initial configuration, but, it was usually close.

For instance one experiment involved 1000 random "inference triangles" involving 3 links, where the nodes were defined to correspond to random subsets of a fixed finite set (so that inheritance probabilities were defined simply in terms of set intersection). Given the specific definition of the random subsets, the mean strength of each of the three inheritance relationships across all the experiments was about .3. The Euclidean distance between the 3-vector of the final (fixed point) link strengths and the 3-vector of the initial link strengths was roughly .075. So the deviation from the true probabilities caused by iterated inference was not very large. Qualitatively similar results were obtained with larger networks.

The key to these experiments is the revision in Step 4: It is assumed that, as iterated inference proceeds, information about the true probabilities is continually merged into the results of inference. If not for this, Step 3 on its own, repeatedly iterated, would lead to noise amplification and increasingly meaningless results. But in a realistic inference context, one would never simply repeat Step 3 on its own. Rather, one would carry out inference on a node or link only when there was new information about that node or link (directly leading to a strength update), or when some new information about other nodes/links indirectly led to inference about that node-link. With enough new information coming in, an inference system has no time to carry out repeated, useless cycles of inference on the same nodes/links – there are always more interesting things to assign resources to. And the ongoing mixing-in of new information about the true strengths with the results of iterated inference prevents the pathologies of circular inference, without the need for a trail mechanism.

What we see from these various experiments is that if one uses an inference control mechanism that avoids the repeated conduction of inference steps in the absence of infusion of new data, issues with circular inference are not severe, and trails are not necessary to achieve reasonable node and link strengths via iterated inference. Circular inference can occur without great harm, so long as one only does it when relevant new data is coming in, or when there is evidence that it is generating information. This is not to say that trail mechanisms are useless in computational systems – they provide an interesting and sometimes important additional layer of protection against circular inference pathologies. But in an inference system that is integrated with an appropriate control mechanism they are not required. The errors induced by circular inference, in practice, may be smaller than many other errors involved in realistic inference. For instance, in the mapping between the brain and uncertain term logic proposed above, we have relied upon a fairly imprecise proportionality between virtual synapse weight and neural inheritance. We are not attempting to argue that the brain implements precise

probabilistic inference, but only an imprecise analogue. Circular inference pathologies are probably not the greatest source of imprecision.

F.3 Neural Representation of Recursion and Abstraction

The material of the previous subsection comprises a speculative but conceptually coherent connection between brain structures and dynamics on the one hand, and probabilistic logic structures and dynamics on the other. However, everything we have discussed so far deals only with first-order term logic, i.e. the logic of inheritance relationships between terms.

Extension to handle similarity relationships, intensional inheritance and so forth is straightforward – but what about more complex term logic constructs, such as would conventionally be expressed using variables and quantifiers. In this section we seek to address this shortcoming, via proposing a hypothesis as to how probabilistic term logic in its full generality might be grounded in neural operations. This material is even more speculative than the above ideas, yet *something* of this nature is critically necessary for completing the conceptual picture.

The handling of quantifiers, in itself, is not the hard part. We have noted above that, in a term logic framework, if one can handle probabilistic variable-bearing expressions and functions, then one handle quantifiers attached to the variables therein. So the essence of the problem is how to handle variables and functions. And we suggest that, when one investigates the issue in detail, a relatively simple hypothesis emerges clearly as essentially the only plausible explanation, if one adopts the neural assembly theory as a working foundational assumption.

In the existing body of mathematical logic and theoretical computer science, there are two main approaches to handling higher-order expressions: variable-based, or combinator-based [CF58, FH98]. It seems highly implausible, to us, that the human brain is implementing some sort of intricate variable-management scheme on the neural-assembly level. Lambda-calculus and other formal schemes for manipulating variables, appear to us to require complexity and precision of a style that self-organizing neural networks are ill-suited to produce via their own style of complex dynamics. Of course it is possible to engineer neural nets that do lambda calculus (as neural nets are Turing-complete), but this sort of neural-net structure seems unlikely to emerge via evolution, and unlikely to get created via known epigenetic processes.

But what about combinators? Here, it seems to us, things are a bit more promising. Combinators are higher-order functions; functions that map functions into functions; functions that map {functions that map functions into functions} into {functions that map functions into functions}, and so forth.

There are specific sets of combinators that are known to give rise to universal computational capability; indeed, there are *many* such specific sets, and one approach to the implementation of functional programming languages is to craft an appropriate set of combinators that combines universality with tractability (the latter meaning, basically, that the combinators have relatively simple definitions; and that pragmatically useful logic expressions tend to have compact representations in terms of the given set of combinators).

We lack the neurodynamic knowledge to say, at this point, that any particular set of combinators seems likely to map into brain function. However, we may still explore the fundamental neural functionalities that would be necessary to give rise to a combinatory-logic-style foundation for abstract neural computation. Essentially, what is needed is the capability to supply one neural assembly as an *input* to another. Note that what we are talking about here is quite different from the standard notion of chaining together neural assemblies, so that the output of assembly A becomes the input of assembly B. Rather, what we are talking about is that *assembly A itself* – as a mapping from inputs to outputs – is fed as an input to assembly B. In this case we may call B a higher-order neural assembly.

Of course, there are numerous possible mechanisms via which higher-order neural assemblies could be implemented in the brain. Here we will discuss just one. Consider a neural assembly A_1 with certain input neurons, certain output neurons, and certain internal "hidden layer" neurons. Then, suppose there exists a "router" neural assembly X, which is at the receiving end of connections from many neurons in A_1, including input, output and hidden neurons. Suppose X is similarly connected to many other neural assemblies: A_2, A_3, ... and so forth; and suppose X contains a "control switch" input that tells it which of these assemblies to pay attention (so, for instance, if the control input is set to 3, then X receives information about A_3). When X is paying attention to a certain assembly, it routes the information it gets from that assembly to its outputs. (Going further, we may even posit a complex control switch, that accepts more involved commands; say, a command that directs the router to a set of k of its input assemblies, and also points it to small neural assembly implementing a combination function that tells it how to combine these k assemblies to produce a composite.)

Finally, suppose the input neurons of assembly B are connected to the router assembly X. Then, depending on how the router switch is set, B may be said to receive one of the assemblies A_k as input. And, next, suppose B's output is directed to the control switch of the router. Then, in effect, B is mapping assemblies to assemblies, in the manner of a higher-order function. And of course, B itself is "just another neural assembly," so that B itself may be routed by the router, allowing for assemblies that map {assemblies mapping assemblies} into assemblies, and so forth.

Where might this kind of "router" assembly exist in the brain? We don't know, at the moment. Quite possibly, the brain may implement higher-order functions by some completely different mechanism. The point we want to

make however, is that there are concrete possibilities via which the brain could implement higher-order logic according to combinatory-logic type mechanisms. Combinators might be neurally represented as neural assemblies interacting with a router assembly, as hypothesized above, and in this way the Hebbian logic mechanisms proposed in the previous sections could be manifested more abstractly, allowing the full-scope of logical reasoning to occur among neural assemblies, with uncertainty management mediated by Hebbian-type synaptic modification.

Appendix G
Possible Worlds Semantics and Experiential Semantics

Co-authored with Matthew Ikle'

G.1 Introduction

The relevance of logic to AGI is often questioned, on the grounds that logic manipulates abstract symbols, but once you've figured out how to translate concrete perception and action into abstract symbols in an appropriate way, you've already solved the hard part of the AGI problem. In this view, human intelligence does logic-like processing as a sort of epiphenomenon, on top of a deeper and more profound layer of sub symbolic processing; and logic is more suitable as a high-level description that roughly approximates the abstract nature of certain thought processes, than as a method of actually realizing these thought processes.

Our own view is that logic is a flexible tool which may be used in many different ways. For example, there is no particular reason not to use logic directly on sensory and actuator data, or on fairly low-level abstractions thereof. This hasn't been the tradition in logic or logic-based AI, but this is a matter of culture and historical accident more than anything else. This would give rise to difficult scalability problems, but so does the application of recurrent neural nets or any other powerful learning approach. In CogPrime we propose to handle the lowest level of sensory and actuator data in a different way, using a CSDLN such as DeSTIN, but we actually believe a PLN approach could be used in place of a system like DeSTIN, without significant loss of efficiency, and only moderate increase in complexity. For example, one could build a CSDLN whose internal operations were all PLN-based – this would make the compositional spatiotemporal hierarchical structure, in effect, into an inference control mechanism.

In this appendix we will explore this region of conceptual space via digging deeper into the semantics of PLN, looking carefully and formally at the connection between PLN terms and relationships, and the concrete experience of an AI system acting in a world. As well as providing a more rigorous foundation for some aspects of the PLN formalism, the underlying concep-

tual purpose is to more fully explicate the relationship between PLN and the world a CogPrime controlled agent lives in.

Specifically, what we treat here is the relation between experiential semantics (on which PLN, and the formal model of intelligent agents presented in Chapter 7 of Part 1, are both founded) and possible-worlds semantics (which forms a more mathematically and conceptually natural foundation for certain aspects of logic, including certain aspects of PLN). In "experiential semantics", the meaning of each logical statement in an agent's memory is defined in terms of the agent's experiences. In "possible worlds semantics", the meaning of a statement is defined by reference to an ensemble of possible worlds including, but not restricted to, the one the agent interpreting the statement has experienced. In this appendix, for the first time, we formally specify the relation between these two semantic approaches, via providing an *experiential grounding of possible worlds semantics*. We show how this simplifies the interpretation of several aspects of PLN, providing a common foundation for setting various PLN system parameters that were previously viewed as distinct.

The reader with a logic background should note that we are construing the notion of possible worlds semantics broadly here, in the philosophical sense [Lew86], rather than narrowly in the sense of Kripke semantics [**?**] and its relatives. In fact there are interesting mathematical connections between the present formulation and Kripke semantics and epistemic logic, but we will leave these for later work.

We begin with indefinite probabilities recalled in Chapter 34, noting that the second-order distribution involved therein may be interpreted using possible worlds semantics. Then we turn to uncertain quantifiers, showing that the third-order distribution used to interpret these in [GIGH08] may be considered as a distribution over possible worlds. Finally, we consider intensional inference, suggesting that the complexity measure involved in the definition of PLN intension [GIGH08] may be derived from a probability measure over possible worlds. The moral of the story is that by considering the space of possible worlds implicit in an agent's experience, one arrives at a simpler unified view of various aspects of the agent's uncertain reasoning, than if one grounds these aspects in the agent's experience directly. This is not an abandonment of experiential semantics but rather an acknowledgement that a simple variety of possible worlds semantics is derivable from experiential semantics, and usefully deployable in the development of uncertain inference systems for general intelligence.

G.2 Inducing a Distribution over Predicates and Concepts

First we introduce a little preliminary formalism. Given a distribution over environments as defined in Chapter 7 of Part 1, and a collection of predicates evaluated on subsets of environments, we will find it useful to define distributions (induced by the distribution over environments) defining the probabilities of these predicates.

Suppose we have a pair (F, T) where F is a function mapping sequences of perceptions into fuzzy truth values, and T is an integer connoting a length of time. Then, we can define the prior probability of (F, T) as the average degree to which F is true, over a random interval of perceptions of length T drawn from a random environment drawn from the distribution over environments. More generally, if one has a pair (F, f), where f is a distribution over the integers, one can define the prior probability of (F, f) as the weighted average of the prior probability of (F, T) where T is drawn from f.

While expressed in terms of predicates, the above formulation can also be useful for dealing with concepts, e.g. by interpreting the concept *cat* in terms of the predicate $isCat$. So we can use this formulation in inferences where one needs a concept probability like $P(cat)$ or a relationship probability like $P(eat(cat, mouse))$.

G.3 Grounding Possible Worlds Semantics in Experiential Semantics

Now we explain how to ground a form of possible worlds semantics in experiential semantics. We explain how an agent, experiencing a single stream of perceptions, may use this to construct an ensemble of possible worlds, which may then be used in various sorts of inferences. This may sound conceptually thorny, but on careful scrutiny it's less so, and in fact is closely related to a commonplace idea in the field of statistics: "subsampling."

The basic idea of subsampling is that, if one has a single dataset D which one wishes to interpret as coming from a larger population of possible datasets, and one wishes to approximately understand the distribution of this larger population, then one can generate a set of additional datasets via removing various portions of D. Each time one removes a certain portion of D, one obtains another dataset, and one can then look at the distribution of these auxiliary datasets, considering it as a model of the population D is drawn from.

This notion ties in closely with the SRAM formal agents model of Chapter 7 of Part 1, which considers a probability distribution over a space of environments which are themselves probability distributions. What a real agent

has is actually a single series of remembered observations. But it can induce a hopeful approximation of this distribution over environments by subsampling its memory and asking: what would it imply about the world if the items in this subsample were the only things I'd seen?

It may be conceptually useful to observe that a related notion to subsampling is found in the literary methodology of science fiction. Many SF authors have followed the methodology of starting with our everyday world, and then changing one significant aspect, and depicting the world as they think it might exist if this one aspect were changed (or, a similar methodology may be followed via changing a small number of aspects). This is a way of generating a large variety of alternate possible worlds from the raw material of our own world.

Applied to SRAM, the subsampling and SF analogies suggest two methods of creating a possible world (and hence, by repetition, an ensemble of possible worlds) from the agent's experience. An agent's interaction sequence with its environment, $ay_{<t} = ay_{1:t-1}$, forms a sample from which it wishes to infer its environment $\mu(y_k|ay_{<k}a_k)$. To better assess this environment, the agent may, for example,

- create a possible world by removing a randomly selected collection of interactions from the agent's memory. In this case, the agent's interaction sequence would be of the form $I_{g,s,t(n_t)} = ay_{(n_t)}$ where (n_t) is some subsequence of $1:t-1$.
- create a possible world via assuming a counterfactual hypothesis (i.e. assigning a statement a truth value that contradicts the agent's experience), and using inference to construct a set of observations that is as similar to its memory as possible, subject to the constraint of being consistent with the hypothesis. The agent's interaction sequence would then look like $bz_{1:t-1}$, where some collection of the $b_k z_k$ differ from the corresponding $a_k y_k$.
- create a possible world by reorganizing portions of the interaction sequence.
- create a possible world by some combination of the above.

We denote an alteration of an interaction sequence $I^a_{g,s,t}$ for an agent a by $\tilde{I}^a_{g,s,t}$, and the set of all such altered interaction sequences for agent a by $\mathcal{I}^a$.

In general, an agent's interaction sequence will presumably be some reasonably likely sequence, and we would therefore be most interested in those cases for which $d_I(I^a_{g,s,t}, \tilde{I}^a_{g,s,t})$ is small, where $d_I(\cdot,\cdot)$ is some measure of sequence similarity, such as neighborhood correlation or PSI-BLAST. The probability distribution ν over environments μ will then tend to give larger probabilities to nearby sequences, as measured by the chosen similarity measure, than to ones that are far away. In colloquial terms, an agent would typically be interested in considering only minor hypothetical changes to its interaction sequences, and would have little basis for understanding the consequences of drastic alterations.

Any of the above methods for altering interaction sequences would alter an agent's perception sequence causing changes to the fuzzy truth values mapped by the function F. This in turn would yield new probability distributions over the space of possible worlds, and thereby yielding altered average probability values for the pair (F, T). This change, constructed from the perspective of the agent based on its experience, could then cause the agent to reassess its action a. Broadly speaking, we call these approaches "experiential possible worlds" or EPW.

The creation of altered interaction sequences may, under appropriate assumptions, provide a basis for creating better estimates for the predicate F than we would otherwise have from a single real-world data point. More specifically we have the following results.

Theorem 1 *Let $\mathcal{E}_n$ represent an arbitrary ensemble of n agents chosen from $\mathcal{A}$. Suppose that, on average over the set of agents $a \in \mathcal{E}_n$, the set of values $F(I)$ for mutated interaction sequences I is normal and unbiased, so that,*

$$E[F] = \frac{1}{n} \sum_{a \in \mathcal{E}_n} \sum_{I^a_{g,s,t} \in \mathcal{I}^a} F(I^a_{g,s,t}) P(I^a_{g,s,t}).$$

Suppose further that these agents explore their environments by creating hypothetical worlds via altered interaction sequences. Then an unbiased estimate for $E[F]$ is given by

$$\hat{F} = \frac{1}{n} \sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}^a_{g,s,t} \in \mathcal{I}^a} F(\tilde{I}^a_{g,s,t}) P(\tilde{I}^a_{g,s,t})$$

$$= \frac{1}{n} \sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}^a_{g,s,t} \in \mathcal{I}^a} F(\tilde{I}^a_{g,s,t}) \sum_{e \in E} [P(e|I^a_{g,s,t}) P(\tilde{I}^a_{g,s,t}|e)].$$

Proof. That $\hat{F}$ is an unbiased estimate for $E[F]$ follows as a direct application of standard statistical bootstrapping theorems. See, for example, [DE96].

Theorem 2 *Suppose that in addition to the above assumptions, we assume that the predicate F is Lipschitz continuous as a function of the interaction sequences $I^a_{g,s,t}$. That is,*

$$d_F \left(F(\tilde{I}^a_{g,s,t}), F(I^a_{g,s,t}) \right) \leq K d_I(\tilde{I}^a_{g,s,t}, I^a_{g,s,t}),$$

for some bound K and $d_F(\cdot, \cdot)$ is a distance measure in predicate space. Then, setting both the bias correction and acceleration parameters to zero, the bootstrap BC_α confidence interval for the mean of F satisfies

$$\hat{F}_{BC_\alpha}[\alpha] \subset [\hat{F} - K z^{(\alpha)} \hat{\sigma_I},\ \hat{F} + K z^{(\alpha)} \hat{\sigma_I}]$$

where $\hat{\sigma_I}$ is the standard deviation for the altered interaction sequences and, letting Φ denote the standard normal c.d.f., $z^{(\alpha)} = \Phi^{-1}(\alpha)$.

Proof. Note that the Lipschitz condition gives

$$\hat{\sigma}_F^2 = \frac{1}{n|\mathcal{I}^a| - 1} \times$$
$$\sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}^a_{g,s,t} \in \mathcal{I}^a} d_F^2 \left(F(\tilde{I}^a_{g,s,t}), F(I^a_{g,s,t}) \right) P(\tilde{I}^a_{g,s,t})$$
$$\leq \frac{K^2}{n|\mathcal{I}^a| - 1} \sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}^a_{g,s,t} \in \mathcal{I}^a} d_I^2(\tilde{I}^a_{g,s,t}, I^a_{g,s,t}) P(\tilde{I}^a_{g,s,t})$$
$$= K^2 \hat{\sigma}_I^2.$$

Since the population is normal and the bias correction and acceleration parameters are both zero, the BC_α bootstrap confidence interval reduces to the standard confidence interval, and the result then follows [**?**].

These two theorems together imply that, on average, through subsampling via altered interaction sequences, agents can obtain unbiased approximations to F and, by keeping the deviations from their experienced interaction sequence small, the deviations of their approximations will also be small.

While the two theorems above demonstrate the power of our subsampling approach, the Lipschitz condition in theorem 2 is a strong assumption. This observation motivates the following modification that is more in keeping with the flavor of PLN's indefinite probabilities approach.

Theorem 3 *Define the set*

$$I^{a;b} = \left\{ \tilde{I}^a_{g,s,t} | d_F^2 \left(F(\tilde{I}^a_{g,s,t}), F(I^a_{g,s,t} \right) = b \right\},$$

and assume that for every real number b the perceptions of the predicate F satisfy

$$\frac{1}{n} \sum_{a \in \mathcal{E}_n} P(I^{a;b}) \leq \frac{M(b)}{b^2} \sigma_I^2$$

for some $M(b) \in \mathbb{R}$. Further suppose that

$$\int_0^1 M(b)\, db = M^2 \in \mathbb{R}.$$

Then under the same assumptions as in Theorem 1, and again setting both the bias correction and acceleration parameters to zero, we have

$$\hat{F}_{BC_\alpha}[\alpha] \subset [\hat{F} - M\sqrt{n} z^{(\alpha)} \widehat{\sigma_I},\ \hat{F} + M\sqrt{n} z^{(\alpha)} \widehat{\sigma_I}]$$

Proof.

$$\hat{\sigma}_F^2 = \frac{1}{n \cdot |\mathcal{I}^a| - 1} \times \sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}^a_{g,s,t} \in \mathcal{I}^a} d_F^2 \left(F(\tilde{I}^a_{g,s,t}), F(I^a_{g,s,t}) \right) P(\tilde{I}^a_{g,s,t})$$

$$= \frac{1}{n \cdot |\mathcal{I}^a| - 1} \times \sum_{a \in \mathcal{E}_n} \int_0^1 \sum_{\tilde{I}^a_{g,s,t} \in \mathcal{I}^{a;b}} d_F^2 \left(F(\tilde{I}^a_{g,s,t}), F(I^a_{g,s,t}) \right) P(\tilde{I}^a_{g,s,t}) \, db$$

$$\leq \frac{1}{n \cdot |\mathcal{I}^a| - 1} \times \sum_{a \in \mathcal{E}_n} \int_0^1 \sum_{\tilde{I}^a_{g,s,t} \in \mathcal{I}^{a;b}} d_F^2 \left(F(\tilde{I}^a_{g,s,t}), F(I^a_{g,s,t}) \right) P(\tilde{I}^a_{g,s,t}) \, db$$

$$\leq b^2 n \frac{M^2}{b^2} \sigma_I^2 = \left(M\sqrt{n} \right)^2 \sigma_I^2.$$

G.4 Reinterpreting Indefinite Probabilities

Indefinite probabilities (see Chapter 34) provide a natural fit with the experiential semantics of the SRAM model, as well as with the subsampling methodology articulated above. A truth-value for an indefinite probability takes the form of a quadruple $([L,\ U],\ b,\ k)$. The meaning of such a truth-value, attached to a statement S is, roughly: There is a probability b that, after k more observations, the truth value assigned to the statement S will lie in the interval $[L,\ U]$. We interpret an interval $[L,\ U]$ by assuming some particular family of distributions (usually Beta) whose means lie in $[L, U]$.

To execute inferences using indefinite probabilities, we make heuristic distributional assumptions, assuming a "first order" distribution of means, with $[L, U\ ]$ as a $(100 \cdot b)\%$ credible interval. Corresponding to each mean in this "first-order" distribution is a "second order" distribution, providing for an "envelope" of distributions.

The resulting bivariate distribution can be viewed as an heuristic approximation intended to estimate unknown probability values existing in hypothetical future situations. Combined with additional parameters, each indefinite truth-value object essentially provides a compact representation of a single second-order probability distribution with a particular, complex structure.

In the EPW context, the second-order distribution in an indefinite probability is most naturally viewed as a distribution over possible worlds; whereas, each first-order distribution represents the distribution of values of the proposition within a given possible world.

As a specific example, consider the case of two virtual agents: one agent, with cat-like characteristics, called "Fluffy" and the second a creature, with dog-like characteristics, named "Muffin." Upon a meeting of the two agents, Fluffy might immediately consider three courses of action: Fluffy might decide to flee as quickly as possible, might hiss and threaten Muffin, or might decide to remain quiet and still. Fluffy might have a memory store of perception sequences from prior encounters with agents with similar characteristics to those of Muffin.

In this scenario, one can view the second-order distribution, as a distribution over all three courses of action that Fluffy might take. Each first-order distribution would then represent the probability distribution of the result from the corresponding action. By hypothetically considering all three possible courses of action and the probability distributions of the resulting action, Fluffy can make more rational decisions even though no result is guaranteed.

G.4.1 Reinterpreting Indefinite Quantifiers

EPW also allows PLN's universal, existential and fuzzy quantifiers to be expressed in terms of implications on fuzzy sets. For example, if we have

ForAll $X
  Implication
    Evaluation F $X
    Evaluation G $X

then this is equivalent to

AverageQuantifier $X
  Implication
    Evaluation F^* $X
    Evaluation G^* $X

where e.g. F^* is the fuzzy set of variations on F constructed by assuming possible errors in the historical evaluations of F. This formulation yields equivalent results to the one given in [GIGH08], but also has the property of reducing quantifiers to FOPLN (over sets derived from special predicates).

To fully understand the equivalence of the above two expressions, first note that in [GIGH08], we handle quantifiers by introducing third-order probabilities. As discussed there, the three levels of distributions are roughly as follows. The first- and second-order levels play the role, with some modifications, of standard indefinite probabilities. The third-order distribution then plays the role of "perturbing" the second-order distribution. The idea is that the second-order distribution represents the mean for the statement $F(x)$.

The third-order distribution then gives various values of F(x) for x, and the first-order distribution gives the sub-distributions for each of the second-order distributions. The final result is then found via an averaging process on all those second-order distributions that are "almost entirely" contained in some $ForAll\_proxy\_interval$.

Next, $AverageQuantifier\ F(\$X)$ is a weighted average of $F(\$X)$ over all relevant inputs $\$X$; and we define the fuzzy set F^* as the set of perturbations of a second-order distribution of hypotheses, and G^* as the corresponding set of perturbed implication results. With these definitions, not only does the above equivalence follow naturally, so do the "possible/perturbed worlds" semantics for the ForAll quantifier. Other quantifiers, including fuzzy quantifiers, can be similarly recast.

G.5 Specifying Complexity for Intensional Inference

A classical dichotomy in logic involves the distinction between extensional inference (which involves sets with members) and intensional inference (which involves entities with properties). In PLN this is handled by taking extension as the foundation (where, in accordance with experiential semantics, sets ultimately boil down to sets of elementary observations), and defining intension in terms of certain fuzzy sets involving observation-sets. This means that in PLN intension, like higher-order inference, ultimately emerges as a subcase of FOPLN (though a subcase with special mathematical properties and special interest for cognitive science and AI). However, the prior formulation of PLN intension contains a "free parameter" (a complexity measure) which is conceptually inelegant; EPW remedies this via providing this parameter with a foundation in possible worlds semantics.

To illustrate how, in PLN, higher-order intensional inference reduces to first-order inferences, consider the case of intensional inheritance. $IntensionalInheritance\ A\ B$ measures the extensional inheritance between the set of properties or patterns associated with A and the corresponding set associated with B. This concept is made precise via formally defining the concept of "pattern," founded on the concept of "association." We formally define the association operator ASSOC through:

```
ExtensionalEquivalence
    Member $E (ExOut ASSOC $C)
    ExOut
            Func
                List
                    ExtensionalInheritance $E $C
                    ExtensionalInheritance
                                            NOT $E
                                            $C
```

where $\text{Func}(x, y) = [x - y]^+$ and $^+$ denotes the positive part.

We next define a pattern in an entity A as something that is associated with, but simpler than, A. Note that this definition presumes some measure $c()$ of complexity. One can then define the fuzzy-set membership function called the "pattern-intensity," via

$$IN(F, G) = [c(G) - c(F)]^+ [P(F|G) - P(F|\neg G)]^+.$$

measuring how much G is a pattern of F. The complexity measure c has been left unspecified in prior explications of PLN, but in the present context we may take it as the measure over concepts implied by the measure over possible worlds derived via subsampling or counterfactuals as described above.

G.6 Reinterpreting Implication between Inheritance Relationships

Finally, one more place where possible worlds semantics plays a role in PLN is with implications such as

Implication
    Inheritance Ben American
    Inheritance Ben obnoxious

We can interpret these by introducing predicates over possible worlds, so that e.g.

$$Z_{Inheritance\_Ben\_American}(W)\langle tv\rangle$$

denotes that tv is the truth value of $Inheritance\_Ben\_American$ in world W. A prerequisite for this, of course, is that Ben and $American$ be defined in a way that spans the space of possible worlds in question. In the case of possible worlds defined by differing subsets of the same observation-set, this is straightforward; in the case of possible worlds defined via counterfactuals it is subtler and we will omit details here.

The above implication may then be interpreted as

AverageQuantifier $W
    Implication
        Evaluation $Z_{Inheritance\_Ben\_American}$ $W
        Evaluation $Z_{Inheritance\_Ben\_obnoxious}$ $W

The weighting over possible worlds $\$W$ may be taken as the one obtained by the system through the subsampling or counterfactual methods as indicated above.

G.7 Conclusion

We began with the simple observation that the mind of an intelligent agent accumulates knowledge based on experience, yet also creates hypothetical knowledge about "the world as it might be," which is useful for guiding future actions. PLN handles this dichotomy via beginning from a foundation in experiential semantics, but then using a form of experientially-grounded possible-worlds semantics to ground a number of particular logical constructs, which we have reviewed here. The technical details we have provided illustrate the general thesis that a combination of experiential and possible-worlds notions may be the best approach to comprehending the semantics of declarative knowledge in generally intelligent agents.

Appendix H
Propositions About Environments in Which CogPrime Components are Useful

H.1 Propositions about MOSES

Why is MOSES a good approach to automated program learning? The conceptual argument in favor of MOSES may be broken down into a series of propositions, which are given here both in informal "slogan" form and in semi-formalized "proposition" form.

Note that the arguments given here appear essentially applicable to other MOSES-related algorithms such as Pleasure as well. The page however was originally written in regard to MOSES and hasn't been revised in the light of the creation of Pleasure.

Slogan 1 refers to "ENF", Elegant Normal Form, which is used by MOSES as a standard format for program trees. This is a way that MOSES differs from GP for example: GP does not typically normalize program trees into a standard syntactic format, but leaves trees heterogeneous as to format.

H.1.1 Proposition: ENF Helps to Guide Syntax-Based Program Space Search

Slogan 1

Iterative optimization is guided based on syntactic distance ==> ENF is good

Proposition 1

On average, over a class C of fitness functions, it is better to do optimization based on a representation in which the (average over all functions in C of the) correlation between syntactic and semantic distance is larger. This should hold for any optimization algorithm which makes a series of guesses, in which the new guesses are chosen from the old ones in a way that is biased to choose new guesses that have small syntactic distance to the old one.

Note that GA, GP, BOA, BOAP and MOSES all fall into the specified category of optimization algorithms

It is not clear what average smoothness condition is useful here. For instance, one could look at the average of d(f(x),f(y))/d(x,y) for d(x,y)<A, where d is syntactic distance and A is chosen so that the optimization algorithm is biased to choose new guesses that have syntactic distance less than A from the old ones.

H.1.2 Demes are Useful if Syntax/Semantics Correlations in Program Space Have a Small Scale

This proposition refers to the strategy of using "demes" in MOSES: instead of just evolving one population of program trees, a collection of "demes" are evolved, each one a population of program trees that are all somewhat similar to each other.

Slogan 2 Small-scale syntactic/semantic correlation ==> demes are good [If the maximal syntactic/semantic correlation occurs on a small scale, then multiple demes are useful]

Proposition 2 Let d denote syntactic distance, and d1 denote semantic distance. Suppose that the correlation between d(x,y) and d1(x,y) is much larger for d(x,y)<A than for A<d(x,y)<2A or A<d(x,y), as an average across all fitness functions in class C. Suppose the number of spheres of radius R required to cover the space of all genotypes is n(R). Then using n(R) demes will provide significantly faster optimization than using n(2R) demes or 1 deme. Assume here the same conditions on the optimization algorithm as in Proposition 1.

Proposition 2.1 Consider the class of fitness functions defined by

```
Correlation( d(x,y), d1(x,y) || d(x,y) = a ) = b
```

Then, across this class, there is a certain number D of demes that will be optimal on average.... I.e. the optimal number of demes depends on the scale-dependence of the correlation between syntactic & semantic distance....

H.1.3 Probabilistic Program Tree Modeling Helps in the Presence of Cross-Modular Dependencies

This proposition refers to the use of BOA-type program tree modeling within MOSES. What it states is that this sort of modeling is useful if the programs in question have significant cross-modular dependences that are not extremely difficult to detect.

Slogan 3 Cross-modular dependencies ==> BOA is good [If the genotypes possess significant internal dependencies that are **not** concordant with the genotypes' internal modular structure, then BOA-type optimization will significantly outperform GA/GP-type optimization for deme-exemplar extension.]

Proposition 3 Consider the classification problem of distinguishing fit genotypes from less fit genotypes, within a deme. If significantly greater classification accuracy can be obtained by classification rules containing "cross-terms" combining genotype elements that are distant from each other within the genotypes - and these cross-terms are not too large relative to the increase in accuracy they provide - then BOA-type modeling will significantly outperform GA/GP-type optimization.

The catch in Proposition 3 is that the BOA-type modeling must be sophisticated enough to recognize the specific cross-terms involved, of course.

H.1.4 Relating ENF to BOA

Now, how does BOA learning relate to ENF?

Proposition 4 ENF decreases, on average, the number and size of cross-terms in the classification rules mentioned in Proposition 3.

H.1.5 Conclusion Regarding Speculative MOSES Theory

What we see from the above is that:

- ENF is needed in order to make the fitness landscape smoother, but can almost never work perfectly so there will nearly always be some long-distance dependencies left after ENF-ization
- The smoother fitness landscape enabled by ENF, enables optimization using demes and incremental exemplar-expansion to work, assuming the number of demes is chosen intelligently
- Within a deme, optimization via incremental exemplar growth is more efficient using BOA than straight evolutionary methods, due to the ability of BOA to exploit the long-distance dependencies not removed by ENF-ization

These propositions appear to capture the basic conceptual justification for the current MOSES methodology. Of course, proving them will be another story, and will likely involve making the proposition statements significantly more technical and complex.

Another interesting angle on these propositions is to view them as constraints on the *problem type* to which MOSES may be fruitfully applied.

Obviously, no program learning algorithm can outperform random search on random program learning problems. MOSES, like any other algorithm, needs to be applied to problems that match its particular biases. What sorts of problems match MOSES's biases?

In particular, the right question to ask is: Given a particular choice regarding syntactic program representation, what sorts of problems match MOSES's biases as induced by this choice?

If the above propositions are correct, the answer is, basically: Problems for which semantic distance (distance in fitness) is moderately well-correlated with syntactic distance (in the chosen representation) over a short scale but not necessarily over a long scale, and for which a significant percentage of successful programs have a moderate but not huge degree of internal complexity (as measured by internal cross-module dependencies).

Implicit in this is an explanation of why MOSES, on its own, is likely not a good approach to solving extremely large and complex problems. This is because for an extremely large and complex problem, the degree of internal complexity of successful programs will likely be too high for BOA modeling to cope with. So then, in these cases MOSES will effectively operate as a multi-start local search on normalized program trees, which is not a stupid thing, but unlikely to be adequately effective for most large, complex problems.

We see from the above that even in the case of MOSES, which is much simpler than OCP, formulating the appropriate theory adequately is not a simple thing, and proving the relevant propositions may be fairly difficult. However, we can also see from the MOSES example that the creation of a theoretical treatment does have some potential for clarifying the nature of the algorithm and its likely range of applicability.

H.2 Propositions About CogPrime

We present some speculations regarding the extension of the approach to MOSES-theory presented above to handle OCP in general. This is of course a much more complex and subtle matter, yet we suggest that in large part it may be handled in a similar way. This way of thinking provides a different perspective on the OCP design - one that has not yet substantially impacted the practical aspects of the design, but may well be of use to us as we iteratively refine the design in the future, in the course of testing and teaching OCP AGI systems.

As with the propositions in previous section but even more so, the details of these heuristic propositions will likely change a fair bit when/if rigorous proof/statement is attempted. But we are intuitively fairly confident that the basic ideas described here will hold up to rigorous analysis.

Finally, one more caveat: the set of propositions listed here is not presented as *complete*. By no means! A complete theoretical treatment of OCP, along

these lines, would involve a more substantial list of related propositions. The propositions given here are meant to cover a number of the most key points, and to serve as illustrations of the sort of AGI theory we believe/hope may be possible to do in the near and medium term future.

H.2.1 When PLN Inference Beats BOA

This proposition explains why, in some cases, it will be better to use PLN rather than BOA within MOSES, for modeling the dependencies within populations of program trees.

Slogan 5 Complex cross-modular dependencies which have similar nature for similar fitness functions ==> PLN inference is better than BOA for controlling exemplar extension

Proposition 5 Consider the classification problem of distinguishing fit genotypes from less fit genotypes, within a deme. If

- significantly greater classification accuracy can be obtained by classification rules containing "cross-terms" combining genotype elements that are distant from each other within the genotypes, but
- the search space for finding these classification rules is tricky enough that a greedy learning algorithm like decision-tree learning (which is used within BOA) isn't going to find the good ones
- the classification rules tend to be similar, for learning problems for which the fitness functions are similar

Then, PLN will significantly outperform BOA for exemplar extension within MOSES, due to its ability to take history into account.

H.2.2 Conditions for the Usefulness of Hebbian Inference Control

Now we turn from MOSES to PLN proper. The approximate probabilistic correctness of PLN is handled via PLN theory itself, as presented in the PLN book. However, the trickiest part of PLN in practice is *inference control*, which in the OCP design is proposed to be handled via "experiential learning." This proposition pertains to the conditions under which Hebbian-style, inductive PLN inference control can be useful.

Slogan 6 If similar theorems generally have similar proofs, then inductively-controlled PLN can work effectively

Proposition 6

- Let L_0 = a simple "base level" theorem-proving framework, with fixed control heuristics

- For $n > 0$, let L_n = theorem-proving done using L_{n-1}, with inference control done using data mining over a DB of inference trees, utilizing L_{n-1} to find recurring patterns among these inference trees that are potentially useful for controlling inference

Then, if T is a set of theorems so that, within T, theorems that are similar according to "similarity provable in L_{n-1} using effort E" have proofs that are similar according to the same measure, then L_n will be effective for proving theorems within T

H.2.3 Clustering-together of Smooth Theorems

This proposition is utilized within Theorem 8, below, which again has to do with PLN inference control.

Slogan 7 "Smooth" theorems tend to cluster together in theorem-space

Proposition 7 Define the smoothness of a theorem as the degree to which its proof is similar to the proofs of other theorems similar to it. Then, smoothness varies smoothly in theorem-space. I.e., a smooth theorem tends to be close-by to other smooth theorems.

H.2.4 When PLN is Useful Within MOSES

Above it was argued that PLN is useful within MOSES due to its capability to take account of history (across multiple fitness functions). But this is not the only reason to utilize PLN within MOSES; Propositions 6 and 7 above give us another theoretical reason.

Proposition 8 If similar theorems of the form "Program A is likely to have similar behavior to program B" tend to have similar proofs, and the conditions of Slogan 6 hold for the class of programs in question, then inductively controlled PLN is good (and better than BOA) for exemplar extension. (This is basically Proposition 6 + Proposition 7)

H.2.5 When MOSES is Useful Within PLN

We have explored theoretical reasons why PLN should be useful within MOSES, as a replacement for the BOA step used in the standalone implementation of MOSES. The next few propositions work in the opposite direction, and explore rasons why MOSES should be useful within PLN, for the specific problem of finding elements of a set given a qualitative (intensional) descrip-

tion of a set. (This is not the only use of MOSES for helping PLN, but it is a key use and a fairly simple one to address from a theoretical perspective.)

Proposition 9 In a universe of sets where intensional similarity and extensional similarity are well-correlated, the problem of finding classification rules corresponding to a set S leads to a population of decently fit candidate solutions with high syntactic/semantic correlation so that demes are good for this problem.

Proposition 10: In a universe of sets satisfying Proposition 9, where sets have properties with complex interdependencies, BOA will be useful for exemplar extension (in the context of using demes to find classification rules corresponding to sets).

Proposition 11: In a universe of sets satisfying Proposition 10, where the interdependencies associated with a set S's property-set vary "smoothly" as S varies, working inference is better than BOA for exemplar extension.

Proposition 12: In a universe of sets satisfying Proposition 10, where the proof of theorems of the form "Both the interdependencies of S's properties, and the interdependencies of T's properties, satisfy predicate F" depends smoothly on the theorem statement, then inductively controlled PLN will be effective for exemplar extension.

H.2.6 On the Smoothness of Some Relevant Theorems

We have talked a bit about smooth theorems, but what sorts of theorems will tend to be smooth? If the OCP design is to work effectively, the "relevant" theorems must be smooth; and the following proposition gives some evidence as to why this may be the case.

Proposition 13 In a universe of sets where intensional similarity and extensional similarity are well-correlated, probabilistic theorems of the form "A is a probabilistic subset of B" and "A is a pattern in B" tend to be smooth....

Note that: For a set S of programs, to say "intensional similarity and extensional similarity are well-correlated" among subsets of S, means the same thing as saying that syntactic and semantic similarity are well-correlated among members of S

Proposition 14 The set of motor control programs, for a set of standard actuators like wheels, arms and legs, displays a reasonable level of correlation between syntactic and semantic similarity

Proposition 15 The set of sentences that are legal in English displays a high level of correlation between syntactic and semantic similarity.

(The above is what, in Chaotic Logic, I called the "principle of continuous compositionality", extending Frege's Principle of Compositionality. It implies that language is learnable via OCP-type methods.... Unlike the other Propositions formulated here, it is more likely to be addressable via statistical than

formal mathematical means; but insofar as English syntax can be formulated formally, it may be considered a roughly-stated mathematical proposition.)

H.2.7 Recursive Use of "MOSES+PLN" to Help With Attention Allocation

Proposition 16 The set of propositions of the form "When thinking about A is useful, thinking about B is often also useful" tends to be smooth - if "thinking" consists of MOSES plus inductively controlled PLN, and the universe of sets is such that this cognitive approach is generally a good one.

This (Prop. 16) implies that adaptive attention allocation can be useful for a MOSES+PLN system, if the attention allocation itself utilizes MOSES+PLN.

H.2.8 The Value of Conceptual Blending

Proposition 17: In a universe of sets where intensional similarity and extensional similarity are well-correlated, if two sets A and B are often useful in proving theorems of the form "C is a (probabilistic) subset of D", then "blends" of A and B will often be useful for proving such theorems as well.

This is a justification of conceptual blending for concept formation.

H.2.9 A Justification of Map Formation

Proposition 18: If a collection of terms A is often used together in MOSES+PLN, then similar collections B will often be useful as well, for this same process ... assuming the universe of sets is so that intensional and extensional similarity are correlated, and MOSES+PLN works well.

This is a partial justification of map formation, in that finding collections B similar to A is achieved by encapsulating A into a node A' and then doing reasoning on A'.

H.3 Concluding Remarks

The above set of propositions is certainly not complete. For instance, one might like to throw in conjunctive pattern mining as a rapid approximation

to MOSES; and some specific justification of artificial economics as a path to effectively utilizing MOSES/PLN for attention allocation; etc.

But, overall, it seems fair to say that the above set of propositions smells like a possibly viable path to a theoretical justification of the OCP design.

To summarize the above ideas in a nutshell, we may say that the effectiveness of the OCP design appears intuitively to follow from the assumptions that:

- within the space of relevant learning problems, problems defined by similar predicates tend to have somewhat similar solutions
- according to OCP's knowledge representation, procedures and predicates with very similar behaviors often have very similar internal structures, and vice versa (and this holds to a drastically lesser degree if the "very" is removed)
- for relevant theorems ("theorems" meaning Atoms whose truth values need to be evaluated, or whose variables or SatisfyingSets need to be filled in, via PLN): similar theorems tend to have similar proofs, and the degree to which this holds varies smoothly in proof-space
- the world can be well modeled using sets for which intensional and extensional similarity are well correlated: meaning that the mind can come up with a system of "extensional categories" useful for describing the world, and displaying characteristic patterns that are not too complex to be recognized by the mind's cognitive methods

To really make use of this sort of theory, of course, two things would need to be done. For one thing, the propositions would have to be proved (which will probably involve some serious adjustments to the proposition statements). For another thing, some detailed argumentation would have to be done regarding why the "relevant problems" confronting an embodied AGI system actually fulfill the assumptions. This might turn out to be the hard part, because the class of "relevant problems" is not so precisely defined. For very specific problems like - to name some examples quasi-randomly - natural language learning, object recogntion, learning to navigate in a room with obstacles, or theorem-proving within a certain defined scope, however, it may be possible to make detailed arguments as to why the assumptions should be fulfilled.

Recall that what makes OCP different from huge-resources AI designs like AIXI (including AIXI^{tl}) and the Gödel Machine is that it involves a number of specialized components, each with their own domains and biases and some with truly general potential as well, hooked together in an integrative architecture designed to foster cross-component interaction and overall synergy and emergence. The strength and weakness of this kind of architecture is that it is specialized to a particular class of environments. AIXI^{tl} and the Gödel Machine can handle any type of environment roughly equally well (which is: very, very slowly), whereas, OCP has the potential to be much faster when it's in environment that poses it learning problems that match

its particular specializations. What we have done in the above series of propositions is to partially formalize the properties an environment must have to be "OCP-friendly." If the propositions are essentially correct, and if interesting real-world environments largely satisfy their assumptions, then OCP is a viable AGI design.

References

AABL02. Nancy Alvarado, Sam S. Adams, Steve Burbeck, and Craig Latta. Beyond the turing test: Performance metrics for evaluating a computer simulation of the human mind. *Development and Learning, International Conf. on*, 0, 2002.

ABR06. M. A. Arbib, J. Bonaiuto, and E. Rosta. The mirror system hypothesis: From a macaque-like mirror system to imitation. In *Proceedings of the 6th International Conference on the Evolution of Language*, pages 3–10. 2006.

ABS+11. Itamar Arel, S Berant, T Slonim, A Moyal, B Li, and K Chai Sim. Acoustic spatiotemporal modeling using deep machine learning for robust phoneme recognition. In *Afeka-AVIOS Speech Processing Conference*, 2011.

Acz88. Peter Aczel. *Non-Well-Founded Sets*. CSLI Press, 1988.

AL03. J. R. Anderson and C. Lebiere. The newell test for a theory of cognition. *Behavioral and Brain Science*, 26, 2003.

All83. James F. Allen. Maintaining knowledge about temporal. *Intervals CACM*, 26:198–3, 1983.

AM01. J. S. Albus and A. M. Meystel. *Engineering of Mind: An Introduction to the Science of Intelligent Systems*. Wiley and Sons, 2001.

Ama85. S. Amari. Differential-geometrical methods in statistics. *Lecture notes in statistics*, 1985.

Ama98. S. Amari. Natural gradient works efficiently in learning. *Neural Computing*, 10:251–276, 1998.

Ami89. Daniel J. Amit. *Modeling brain function – the world of attractor neural networks*. Cambridge University Press, New York, USA, 1989.

AN00. Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*. AMS, 2000.

ARC09. I. Arel, D. Rose, and R. Coop. Destin: A scalable deep learning architecture with application to high-dimensional robust pattern recognition. *Proc. AAAI Workshop on Biologically Inspired Cognitive Architectures*, 2009.

ARK09a. I. Arel, D. Rose, and T. Karnowski. A deep learning architecture comprising homogeneous cortical circuits for scalable spatiotemporal pattern inference. *NIPS 2009 Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

Ark09b. Ronald Arkin. *Governing Lethal Behavior in Autonomous Robots*. Chapman and Hall, 2009.

Arl75. P. K. Arlin. *Cognitive development in adulthood: A fifth stage?*, volume 11. Developmental Psychology, 1975.

Arn69. Rudolf Arnheim. *Visual Thinking*. University of California Press. Berkeley, 1969.

AS94. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases*, 1994.

Aus99. James Austin. *Zen and the Brain*. MIT Press, 1999.

Bac09. Joscha Bach. *Principles of Synthetic Intelligence*. Oxford University Press, 2009.

Bar89. Jon Barwise. *The Situation in Logic*. CLSI Press, 1989.

Bat79. Gregory Bateson. *Mind and Nature: A Necessary Unity*. New York: Ballantine, 1979.

Bau04. E. B. Baum. *What is Thought?* MIT Press, 2004.

Bau06. E. Baum. A working hypothesis for general intelligence. In *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms*, 2006.

BBF+01. G. Buccino, F. Binkofski, G. R. Fink, L. Fadiga, L. Fogassi, V. Gallese, R. J. Seitz, K. Zilles, G. Rizzolatti, and H. J. Freund. Action observation activates premotor and parietal areas in a somatotopic manner: an fMRI study. *European Journal of Neuroscience*, 13:400–404, 2001.

BDL93. Louise Barrett, Robin Dunbar, and John Lycett. *Human Evolutionary Psychology*. Princeton University Press, 1993.

BE89. Jon Barwise and John Etchemendy. *The Liar: An Essay on Truth and Circularity*. Oxford University Press, 1989.

Ben94. Brandon Bennett. Spatial reasoning with propositional logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 4th International Conference (KR94)*, pages 51–62. Morgan Kaufmann, 1994.

BF71. J. D. Bransford and J. Franks. The abstraction of linguistic ideas. *Cognitive Psychology*, 2:331–350, 1971.

BF97. A Blum and M Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 1997.

BF09. Bernard Baars and Stan Franklin. Consciousness is computational: The lida model of global workspace theory. *International Journal of Machine Consciousness.*, 2009.

bGBK02.

1. Goertzel, Andrei Klimov Ben, and Arkady Klimov. Supercompiling java programs, 2002.

BH05. Sebastian Bader and Pascal Hitzler. Dimensions of neural-symbolic integration - a structured survey. In S. Artemov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb, and J. Woods., editors, *We Will Show Them: Essays in Honour of Dov Gabbay*, volume 1, pages 167–194. College Publications, 2005.

BH10. Bundzel and Hashimoto. Object identification in dynamic images based on the memory-prediction theory of brain function. *Journal of Intelligent Learning Systems and Applications*, 2-4, 2010.

Bic88. M. Bickhard. Piaget on variation and selection models: Structuralism, logical necessity, and interactivism. *Human Development*, 31:274–312, 1988.

Bic08. Derek Bickerton. *Bastard Tongues*. Hill and Wang, 2008.

Bil05. Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337:2005, 2005.

BKL06. Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proc. International Conference on Machine Learning*, 2006.

BL99. Avrim Blum and John Langford. Probabilistic planning in the graphplan framework. In *5th European Conference on Planning (ECP '99)*, 1999.

Bla06. Sandra Blakeslee. Cells that Read Minds. *New York Times*, Jan 10 2006.

BLC+04. G. Buccino, F. Lui, N. Canessa, I. Patteri, G. Lagravinese, F. Benuzzi, C. A. Porro, and G. Rizzolatti. Neural circuits involved in the recognition of actions performed by nonconspecifics: An fMRI study. *J. Cogn. Neurosci.*, 16:114–126, 2004.

BO09. A. Baranes and Pierre-Yves Oudeyer. R-iac: Robust intrinsically motivated active learning. *Proc. of the IEEE International Conf. on Learning and Development, Shanghai, China.*, 33, 2009.

Bol98. B. Bollobas. *Modern Graph Theory*. Springer, 1998.

Bor05. Christian Borgelt. Keeping things simple: Finding frequent item sets by recursive elimination. In *Workshop on Open Source Data Mining Software (OSDM'05). Chicago IL*, pages 66–70. 2005.

Bos02. Nick Bostrom. Existential risks. *Journal of Evolution and Technology*, 9, 2002.

Bos03. Nick Bostrom. Ethical issues in advanced artificial intelligence. In Iva Smit, editor, *Cognitive, Emotive and Ethical Aspects of Decision Making in Humans and in Artificial Intelligence*, volume 2., pages 12–17. 2003.

Bro60. James Cooke Brown. *Loglan*. Scientific American, June 1960.

Bro84. J. Broughton. Not beyond formal operations, but beyond piaget. In M. Commons F. Richards and C. Armon, editors, *Beyond Formal Operations: Late Adolescent and Adult Cognitive Development*, pages 395–411. Praeger. New York, 1984.

BS04. B. Bakker and Juergen Schmidhuber. Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization. *Proc. of the 8-th Conf. on Intelligent Autonomous Systems*, 2004.

BW88. R. W. Byrne and A. Whiten. *Machiavellian Intelligence*. Clarendon Press, 1988.

BZGS06. B. Bakker, V. Zhumatiy, G. Gruener, and Juergen Schmidhuber. Quasi-online reinforcement learning for robots. *Proc. of the International Conf. on Robotics and Automation*, 2006.

Cal96. William Calvin. *The Cerebral Code*. MIT Press, 1996.

Car85. S. Carey. *Conceptual Change in Childhood*. MIT Press, 1985.

Car06. Pereira Francisco Cara. *Creativity and Artificial Intelligence: A Conceptual Blending Approach, Applications of Cognitive Linguistics*. Amsterdam: Mouton de Gruyter, 2006.

Cas85. R. Case. *Intellectual development: Birth to adulthood*. Academic Press, 1985.

Cas04. N. L. Cassimatis. Grammatical processing using the mechanisms of physical inferences. In *Proceedings of the Twentieth-Sixth Annual Conference of the Cognitive Science Society*. 2004.

Cas07. Nick Cassimatis. Adaptive algorithmic hybrids for human-level artificial intelligence. 2007.

CB00. W. H. Calvin and D. Bickerton. *Lingua ex Machina*. MIT Press, 2000.

CF58. H. B. Curry and R. Feys. *Combinatory Logic*. North-Holland, Amsterdam, Holland, 1958.

CFH97. Eliseo Clementini, Paolino Di Felice, and Daniel HernÃ¡ndez. Qualitative representation of positional information. *Artificial Intelligence*, 95:317–356, 1997.

Cha08. Gregory Chaitin. *Algorithmic Information Theory*. Cambridge University Press, 2008.

Cha09. Mark Changizi. *The Vision Revolution*. BenBella Books, 2009.

Che97. K. Chellapilla. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1997.

CMH+07. T. F. T. Collins, E. O. Mann, M. R. H. Hill, E. J. Dommett, and S. A. Greenfielddynamics of neuronal assemblies are modulated by anaesthetics but not analgesics. *Eur J Anaesthesiol.*, 24–7, 2007.

Coh95. A.G. Cohn. A hierarchical representation of qualitative shape based on connection and convexity. In *Proc COSIT95, LNCS*, pages 311–326. Springer Verlag, 1995.

Cow97. John Woldemar Cowan. *The Complete Lojban Language*. The Logical Language Group, 1997.

CP05. M. L. Commons and A. Pekker. Hierarchical complexity: A formal theory. http://www.dareassociation.org/Papers/Hierarchical%20Complexity%20-%20A%20Formal%20Theory%20(Commons%20&%20Pekker).pdf, 2005.

CRK82. M Commons, F Richards, and D Kuhn. Systematic and metasystematic reasoning: a case for a level of reasoning beyond Piaget's formal operations. *Child Development. 53*, 53:1058–1069, 1982.

CSZ06. Olivier Chapelle, Bernhard Schakopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006.

CTS+98. M. Commons, E. J. Trudeau, S. A. Stein, F. A. Richards, and S. R. Krause. Hierarchical complexity of tasks shows the existence of developmental stages. *Developmental Review. 18*, 18.:237–278, 1998.

Dab99. A.G. Dabak. *A Geometry for Detection Theory*. PhD Thesis, Rice U., 1999.

Dam00. Antonio Damasio. *The Feeling of What Happens*. Harvest Books, 2000.

Dav84. D. Davidson. *Inquiries into Truth and Interpretation*. Oxford: Oxford University Press, 1984.

DE96. Thomas J. DiCiccio and Bradley Efron. *Bootstrap confidence intervals*, volume 11–3. Statistical Science, 1996.

Den87. D. Dennett. *The Intentional Stance*. Cambridge, MA: MIT Press, 1987.

Den91. Daniel Dennett. *Brainstorms*. Cambridge MA: MIT Press, 1991.

DG94. Dixon and G. *Division Algebras: Octonions, Quaternions, Complex Numbers and the Algebraic Design of Physics*. Kluwer, 1994.

DG05. Hugo De Garis. *The Artilect War*. ETC, 2005.

DOP08. Wlodzislaw Duch, Richard Oentaryo, and Michel Pasquier. Cognitive architectures: Where do we go from here? *Proc. of the Second Conf. on AGI*, 2008.

DP09. Yassine Djouadi and Henri Prade. Interval-valued fuzzy formal concept analysis. In *ISMIS '09: Proc. of the 18th International Symposium on Foundations of Intelligent Systems*, pages 592–601, Berlin, Heidelberg, 2009. Springer-Verlag.

Dör02. Dietrich Dörner. *Die Mechanik des Seelenwagens. Eine neuronale Theorie der Handlungsregulation*. Verlag Hans Huber, 2002.

EBJ+97. J. Elman, E. Bates, M. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett. *Rethinking Innateness: A Connectionist Perspective on Development*. MIT Press, 1997.

Ede93. Gerald Edelman. Neural darwinism: Selection and reentrant signaling in higher brain function. *Neuron*, 10, 1993.

Elm91. J. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–226, 1991.

FB82. Fuller and Buckminster. *Synergetics*. Macmillan Publishing, 1982.

FB08. Stan Franklin and Bernard Baars. Possible neural correlates of cognitive processes and modules from the lida model of cognition. *Cognitive Computing Research Group, University of Memphis*, 2008. http://ccrg.cs.memphis.edu/tutorial/correlates.html.

FC86. R. Fung and C. Chong. Metaprobability and Dempster-shafer in evidential reasoning. In L. Kanal and J. Lemmer. North-Holland, editors, *Uncertainty in Artificial Intelligence*, pages 295–302. 1986.

FF92. Christian Freksa and Robert Fulton. Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1-2):199 – 227, 1992.

FH98. A. J. Field and P. G. Harrison. *Functional Progamming*. Addison-Wesley, Boston, MA, 1998.

Fis80. K. Fischer. A theory of cognitive development: control and construction of hierarchies of skills. *Psychological Review*, 87:477–531, 1980.

FL12. Jeremy Fishel and Gerald Loeb. Bayesian exploration for intelligent identification of textures. *Frontiers in Neurorobotics 6-4*, 2012.

Fod94. J. Fodor. *The Elm and the Expert*. Cambridge, MA: Bradford Books, 1994.

Fra06. Stan Franklin. The lida architecture: Adding new modes of learning to an intelligent, autonomous, software agent. *Int. Conf. on Integrated Design and Process Technology*, 2006.

Fri98. Roy. Frieden. *Physics from Fisher Information*. Cambridge U. Press, 1998.

FT02. G. Fauconnier and M. Turner. *The Way We Think: Conceptual Blending and the Mind's Hidden Complexities*. Basic, 2002.

GBK04. S. Gustafson, E. K. Burke, and G. Kendall. Sampling of unique structures and behaviours in genetic programming. In *European Conf. on Genetic Programming*, 2004.

GCG+11. Ben Goertzel, Lucio Coelho, Nil Geisweiller, Predrag Janicic, and Cassio Pennachin. *Real World Reasoning*. Atlantis Press, 2011.

GCPM06. Ben Goertzel, Lucio Coelho, Cassio Pennachin, and Mauricio Mudada. Identifying Complex Biological Interactions based on Categorical Gene Expression Data. In *Proceedings of Conference on Evolutionary Computing. Vancouver CA*. 2006.

GD09. Ben Goertzel and Deborah Duong. Opencog ns: An extensible, integrative architecture for intelligent humanoid robotics. 2009.

GdG08. Ben Goertzel and Hugo de Garis. Xia-man: An extensible, integrative architecture for intelligent humanoid robotics. pages 86–90, 2008.

GE86. R. Gelman and E. Meck and s. *Merkin (1986). Young children's numerical competence. Cognitive Development*, 1:1–29, 1986.

GE01. Roop Goyal and Max Egenhofer. Similarity in cardinal directions. In *In Proc. of the Seventh International Symposium on Spatial and Temporal Databases*, pages 36–55. Springer-Verlag, 2001.

GEA08. Ben Goertzel and Cassio Pennachin Et Al. An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in second life. In *Proc.of the First Conf. on AGI*. IOS Press, 2008.

GGK. Thomas Gilovich, Dale Griffin, and Dale Kahneman. *Heuristics and Biases*. Cambridge University Press.

GH09. Dileep George and Jeff Hawkins. Towards a mathematical theory of cortical micro-circuits. *PLoS Comput Biol 5*, 2009.

GH11. N Garg and J Henderson. Temporal restricted boltzmann machines for dependency parsing. In *Proc. ACL*, 2011.

GI11. B. Goertzel and M. Iklé. Steps toward a geometry of mind. In J Schmidhuber and K Thorisson, editors, *Subm.to AGI-11*. Springer, 2011.

Gib77. J. J. Gibson. The theory of affordances. In R. Shaw & J. Bransford. Erlbaum, editor, *Perceiving, Acting and Knowing*. 1977.

Gib78. John Gibbs. Kohlberg's moral stage theory: a Piagetian revision. *Human Development*, 22:89–112, 1978.

Gib79. J. J. Gibson. *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin, 1979.

GIGH08. B. Goertzel, M. Ikle, I. Goertzel, and A. Heljakka. *Probabilistic Logic Networks*. Springer, 2008.

Gil82. Carol Gilligan. *In a Different Voice*. Cambridge, MA: Harvard University Press, 1982.

GKD89. D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 1989.

GL10. Ben Goertzel and Ruiting Lian. A probabilistic characterization of fuzzy semantics. *Proc. of ICAI-10, Beijing*, 2010.

GLdG$^+$10. Ben Goertzel, Ruiting Lian, Hugo de Garis, Shuo Chen, and Itamar Arel. World survey of artificial brains, part ii: Biologically inspired cognitive architectures. *Neurocomputing*, April 2010.

GMIH08. B. Goertzel, I. Goertzel M. Ikle, and A. Heljakka. *Probabilistic Logic Networks*. Springer, 2008.

GN02. Alfonso Gerevini and Bernhard Nebel. Qualitative spatio-temporal reasoning with rcc-8 and allen's interval calculus: Computational complexity. In Frank van Harmelen, editor, *ECAI*, pages 312–316. IOS Press, 2002.

Goe94. Ben Goertzel. *Chaotic Logic*. Plenum, 1994.

Goe01. Ben Goertzel. *Creating Internet Intelligence*. Plenum Press, 2001.

Goe06a. Ben Goertzel. *The Hidden Pattern*. Brown Walker, 2006.

Goe06b. Ben Goertzel. *The Hidden Pattern*. Brown Walker, 2006.

Goe08a. B. Goertzel. The pleasure algorithm. *groups.google.com/group/opencog/files*, 2008.

Goe08b. Ben Goertzel. A pragmatic path toward endowing virtually-embodied ais with human-level linguistic capability. IEEE World Congress on Computational Intelligence (WCCI), 2008.

Goe09a. Ben Goertzel. Cognitive synergy: A universal principle of feasible general intelligence? 2009.

Goe09b. Ben Goertzel. Opencog prime: A cognitive synergy based architecture for embodied artificial general intelligence. In *ICCI 2009, Hong Kong*, 2009.

Goe10a. Ben Goertzel. Coherent aggregated volition. *Multiverse According to Ben*, 2010. http://multiverseaccordingtoben.blogspot.com/2010/03/coherent-aggregated-volition-toward.htm.

Goe10b. Ben Goertzel. Infinite-order probabilities and their application to modeling self-referential semantics. *Proc. of ICAI-10, Beijing*, 2010.

Goe10c. Ben et al Goertzel. A general intelligence oriented architecture for embodied natural language processing. In *Proc. of the Third Conf. on Artificial General Intelligence (AGI-10)*. Atlantis Press, 2010.

Goe10d. Ben et al Goertzel. Opencogbot: An integrative architecture for embodied agi. *Proc. of ICAI-10, Beijing*, 2010.

Goe11a. B Goertzel. Integrating a compositional spatiotemporal deep learning network with symbolic representation/reasoning within an integrative cognitive architecture via an intermediary semantic network. In *Proceedings of AAAI Symposium on Cognitive Systems,*, 2011.

Goe11b. Ben Goertzel. Imprecise probability as a linking mechanism between deep learning, symbolic cognition and local feature detection in vision processing. In *Proc. of AGI-11*, 2011.

Goeon. Ben Goertzel. Perception processing for general intelligence, part i: Representationally transparent deep learning. (in preparation).

Goo86. I. Good. *The Estimation of Probabilities*. Cambridge, MA: MIT Press, 1986.

Gor86. R. Gordon. Folk psychology as simulation. *Mind and Language. 1*, 1.:158–171, 1986.

GPI+10. Ben Goertzel, Joel Pitt, Matthew Ikle, Cassio Pennachin, and Rui Liu. Glocal memory: a design principle for artificial brains and minds. *Neurocomputing*, April 2010.

GPPG06. Ben Goertzel, Hugo Pinto, Cassio Pennachin, and Izabela Freire Goertzel. Using dependency parsing and probabilistic inference to extract relationships between genes, proteins and malignancies implicit among multiple biomedical research abstracts. In *Proc. of Bio-NLP 2006*, 2006.

GPW+11. Ben Goertzel, Joel Pitt, Jared Wigmore, Nil Geisweiller, Zhenhua Cai, Ruiting Lian, Deheng Huang, and Gino Yu. Cognitive synergy between procedural and declarative learning in the control of animated and robotic agents using the opencogprime agi architecture. In *Proceedings of AAAI-11*, 2011.

GR00. Alfonso Gerevini and Jochen Renz. Combining topological and size information for spatial reasoning. *Artificial Intelligence*, 137:2002, 2000.

Gre01. Susan Greenfield. *The Private Life of the Brain*. Wiley, 2001.

GSW05. Bernhard Ganter, Gerd Stumme, and Rudolf Wille. *Formal Concept Analysis: Foundations and Applications*. Springer-Verlag, 2005.

Ham87. Stuart Hameroff. *Ultimate Computing*. North Holland, 1987.

Ham10. Stuart Hameroff. The Òconscious pilotÓÑdendritic synchrony moves through the brain to mediate consciousness. *Journal of Biological Physics*, 2010.

HB06. Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Brown Walker, 2006.

HDY+12. Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Tara Sainathand Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 2012.

Heb49. Donald Hebb. *The organization of behavior*. Wiley, 1949.

Hey07. F. Heylighen. *The Global Superorganism: an evolutionary-cybernetic model of the emerging network society*. Social Evolution and History 6-1, 2007.

HG08. David Hart and Ben Goertzel. Opencog: A software framework for integrative artificial general intelligence. In *AGI*, volume 171 of *Frontiers in Artificial Intelligence and Applications*, pages 468–472. IOS Press, 2008.

HH07. Barbara Hammer and Pascal Hitzler, editors. *Perspectives of Neural-Symbolic Integration. Studies in Computational Intelligence, Vol. 77.* Springer, 2007.

Hib02. Bill Hibbard. *Superintelligent Machines.* Springer, 2002.

HK02. David Harel and Yehuda Koren. *Graph Drawing by High-Dimensional Embedding.* 2002.

Hob78. J. Hobbs. Resolving pronoun references. *Lingua*, page 311Ð338, 1978.

Hof79. Douglas Hofstadter. *Godel, Escher, Bach: An Eternal Golden Braid.* Basic, 1979.

Hof96. Douglas Hofstadter. *Fluid Concepts and Creative Analogies.* Basic Books, 1996.

Hol75. J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

Hop82. J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. of the National Academy of Sciences*, 79:2554–2558, 1982.

HOT06. G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

HP91. Gerard Huet and G. Plotkin. *Logical Frameworks.* Cambridge University Press, 1991.

Hud90. Richard Hudson. *English Word Grammar.* Blackwell Press, 1990.

Hud07. Richard Hudson. *Language Networks. The new Word Grammar.* Oxford University Press, 2007.

Hut95. E. Hutchins. *Cognition in the Wild.* Bradford, 1995.

Hut96. Edwin Hutchins. *Cognition in the Wild.* MIT Press, 1996.

Hut99. G. Hutton. A tutorial on the universality and expressiveness of fold. *Journal of Functional Programming*, 1999.

Hut05. Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability.* Springer, 2005.

HWP03. Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, pages 549–552. 2003.

HZT+02. J. Han, S. Zeng, K. Tham, M. Badgero, and J. Weng. Dav: A humanoid robot platform for autonomous mental development,. *Proc. 2nd International Conf. on Development and Learning*, 2002.

IP58. B. Inhelder and J. Piaget. *The Growth of Logical Thinking from Childhood to Adolescence.* Basic Books, 1958.

Jac03. Ray Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution.* Oxford University Press, 2003.

JL08. D. J. Jilk and C. Lebiere. and o'reilly. *R. C. and Anderson, J. R. (2008). SAL: An explicitly pluralistic cognitive architecture. Journal of Experimental and Theoretical Artificial Intelligence*, 20:197–218, 2008.

JM09. Daniel Jurafsky and James Martin. *Speech and Language Processing.* Pearson Prentice Hall, 2009.

Joh05. Mark Johnson. *LaTeX: A Developmental Cognitive Neuroscience.* Wiley-Blackwell, 2005.

Joy00. Bill Joy. *Why the future doesn't need us, Wired.* April 2000.

KA95. J. R. Koza and D. Andre. Parallel genetic programming on a network of transputers. Technical report, Stanford University, 1995.

Kam91. George Kampis. *Self-Modifying Systems in Biology and Cognitive Science.* Plenum Press, 1991.

Kan64. Immanuel Kant. *Groundwork of the Metaphysic of Morals.* Harper and Row, 1964.

KAR10. Tom Karnowski, Itamar Arel, and D. Rose. Deep spatiotemporal feature learning with application to image classification. In *The 9th International Conference on Machine Learning and Applications (ICMLA'10)*, 2010.

Kau. Louis Kauffmann. *Sign and Space*. Louis Kauffmann.

KE06. J. L. Krichmar and G. M. Edelman. Principles underlying the construction of brain-based devices. In T. Kovacs and J. A. R. Marshall, editors, *Adaptation in Artificial and Biological Systems*, pages 37–42. 2006.

KK90. K. Kitchener and P. King. Reflective judgement: ten years of research. In M. Commons. Praeger. New York, editor, *Beyond Formal Operations: Models and Methods in the Study of Adolescent and Adult Thought*, volume 2, pages 63–78. 1990.

KK01. Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 313–320. 2001.

Koz92. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

Koz94. J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

Kur06. Ray Kurzweil. *The Singularity is Near*. 2006.

Kur09. Yohei Kurata. 9-intersection calculi for spatial reasoning on the topological relations between multi-domain objects. USA, June 2009.

Kur12. Ray Kurzweil. *How to Create a Mind*. Viking, 2012.

LA93. C Lebiere and J R Anderson. A connectionist implementation of the act-r production system. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, 1993.

Lai72. R.D. Laing. *Knots*. Vintage, 1972.

Lai12. John E Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.

Lan05. Pat Langley. An adaptive architecture for physical agents. *Proc. of the 2005 IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology*, 2005.

LAon. C. Lebiere and J. R. Anderson. The case for a hybrid architecture of cognition. (in preparation).

LBDE90. Y. LeCun, B. Boser, J. S. Denker, and Al. Et. Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems*, 2, 1990.

LBH10. Jens Lehmann, Sebastian Bader, and Pascal Hitzler. Extracting reduced logic programs from artificial neural networks. *Applied Intelligence*, 2010.

LD03. A. Laud and G. Dejong. The influence of reward on the speed of reinforcement learning. *Proc. of the 20th International Conf. on Machine Learning*, 2003.

Leg06a. Shane Legg. Friendly ai is bunk. *Vetta Project*, 2006. http://commonsenseatheism.com/wp-content/uploads/2011/02/Legg-Friendly-AI-is-bunk.pdf.

Leg06b. Shane Legg. Unprovability of friendly ai. *Vetta Project*, 2006. http://www.vetta.org/2006/09/unprovability-of-friendly-ai/.

Lem10. B. Lemoine. Nlgen2: a linguistically plausible, general purpose natural language generation system. http://www.louisiana.edu/??bal2277/NLGen2, 2010.

Lev66. VI Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady 10*, page 707Ð10., 1966.

Lev94. L. Levin. Randomness and nondeterminism. In *The International Congress of Mathematicians*, 1994.

Lew86. David Lewis. *On the Plurality of Worlds*. Basil Blackwell, 1986.

LG90. Douglas Lenat and R. V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, 1990.

LG09. Moshe Looks and Ben Goertzel. Program representation for general intelligence. *Proc. of AGI-09*, 2009.

LGE10. Ruiting Lian, Ben Goertzel, and Al Et. Language generation via glocal similarity matching. *Neurocomputing*, 2010.

LH07. Shane Legg and Marcus Hutter. A collection of definitions of intelligence. IOS, 2007.

LKP+05. Sung Hee Lee, Junggon Kim, Frank Chongwoo Park, Munsang Kim, and James E. Bobrow. Newton-type algorithms for dynamics-based robot movement optimization. *IEEE Transactions on Robotics*, 21(4):657–667, 2005.

LLR09. Weiming Liu, Sanjiang Li, and Jochen Renz. Combining rcc-8 with qualitative direction calculi: Algorithms and complexity. In *IJCAI*, 2009.

LLW+05. Guang Li, Zhengguo Lou, Le Wang, Xu Li, and Walter J Freeman. Application of chaotic neural model based on olfactory system on pattern recognition. *ICNC*, 1:378–381, 2005.

LMC07a. M. H. Lee, Q. Meng, and F. Chao. Developmental learning for autonomous robots. *Robotics and Autonomous Systems*, 2007.

LMC07b. M. H. Lee, Q. Meng, and F. Chao. Staged competence learning in developmental robotics. *Adaptive Behavior*, 2007.

LMDK07. Thomas K. Landauer, Danielle S. McNamara, Simon Dennis, and Walter Kintsch. *Handbook of Latent Semantic Analysis*. Psychology Press, 2007.

LN00. George Lakoff and Rafael Nunez. *Where Mathematics Comes From*. Basic Books, 2000.

Log07. Robert M. Logan. *The Extended Mind*. University of Toronto Press, 2007.

Loo06. Moshe Looks. *Competent Program Evolution*. PhD Thesis, Computer Science Department, Washington University, 2006.

Loo07a. M. Looks. On the behavioral diversity of random programs. In *Genetic and evolutionary computation conference*, 2007.

Loo07b. M. Looks. Scalable estimation-of-distribution program evolution. In *Genetic and evolutionary computation conference*, 2007.

Low99. David Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conf. on Computer Vision*, pages 1150–1157, 1999.

LP02. W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

LRN87. John Laird, Paul Rosenbloom, and Alan Newell. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33, 1987.

LWML09. John Laird, Robert Wray, Robert Marinier, and Pat Langley. Claims and challenges in evaluating human-level intelligent systems. *Proc. of AGI-09*, 2009.

Mac95. D. MacKenzie. The automation of proof: A historical and sociological exploration. *IEEE Annals of the History of Computing*, 17(3):7–29, 1995.

Mai00. Monika Maidl. The common fragment of ctl and ltl. In *In IEEE Symposium on Foundations of Computer Science*, pages 643–652, 2000.

Mar01. H. Marchand. *Reflections on PostFormal Thought*. The Genetic Epistemologist, 2001.

May04. M. T. Maybury. *New Directions in Question Answering*. MIT Press, 2004.

McK03. Bill McKibben. *Enough: Staying Human in an Engineered Age*. Saint Martins Griffin, 2003.

Met04. Thomas Metzinger. *Being No One*. Bradford, 2004.

Mih07. Rada Mihalcea. *Word sense disambiguation. Encyclopedia of Machine Learning*. Springer-Verlag, 2007.

Min88. Marvin Minsky. *The Society of Mind*. MIT Press, 1988.

Min07. Marvin Minsky. *The Emotion Machine*. 2007.

MK08. Jonathan Mugan and Benjamin Kuipers. Towards the application of reinforcement learning to undirected developmental learning. *International Conf. on Epigenetic Robotics*, 2008.

MK09. Jonathan Mugan and Benjamin Kuipers. Autonomously learning an action hierarchy using a learned qualitative state representation. *IJCAI-09*, 2009.

MS99. Christopher Manning and Heinrich Scheutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

MSV+08. G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori. The icub humanoid robot: an open platform for research in embodied cognition. *Performance Metrics for Intelligent Systems Workshop (PerMIS 2008)*, 2008.

MTW73. C Misner, K Thorne, and J Wheeler. *Gravitation*. Freeman, 1973.

MW83. Franklin Merrell-Wolf. *The Philosophy of Consciousness Without an Object*. Three Rivers Press, 1983.

Nan08. Nanowerk. Carbon nanotube rubber could provide e-skin for robots. http://www.nanowerk.com/news/newsid=6717.php, 2008.

NK04. A. Nestor and B. Kokinov. Towards active vision in the dual cognitive architecture. *International Journal on Information Theories and Applications*, 11, 2004.

OCC90. Andrew Ortony, Gerald Clore, and Allan Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, 1990.

Ols95. J. R. Olsson. Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 1995.

Omo08. Stephen Omohundro. The basic ai drives. Proceedings of the First AGI Conference. IOS Press, 2008.

Omo09. Stephen Omohundro. Creating a cooperative future. 2009. http://selfawaresystems.com/2009/02/23/talk-on-creating-a-cooperative-future/.

Opa52. A. I. Oparin. *The Origin of Life*. Dover, 1952.

PAF00. H. Park, S. Amari, and K. Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Computing*, 13:755–764, 2000.

Pal04. Girish Keshav Palshikar. Fuzzy region connection calculus in finite discrete space domains. *Appl. Soft Comput.*, 4(1):13–23, 2004.

PCP00. Papageorgiou, C., and T. Poggio. *A trainable system for object detection*, volume 38–1. Intl. J ComputerVision, 2000.

Pei34. C. Peirce. *Collected papers: Volume V. Pragmatism and pragmaticism*. Harvard University Press. Cambridge MA., 1934.

Pel05. Martin Pelikan. *Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms*. Springer, 2005.

Pen96. Roger Penrose. *Shadows of the Mind*. Oxford University Press, 1996.

Per70. William G. Perry. *Forms of Intellectual and Ethical Development in the College Years: A Scheme*. Holt, Rinehart and Winston, 1970.

Per81. William G. Perry. Cognitive and ethical growth: The making of meaning. In Arthur W. Chickering. Jossey-Bass. San Francisco, editor, *The Modern American College*, pages 76–116. 1981.

Pia53. Jean Piaget. *The Origins of Intelligence in Children*. Routledge and Kegan Paul, 1953.

Pia55. Jean Piaget. *The Construction of Reality in the Child*. Routledge and Kegan Paul, 1955.

Pir84. Robert Pirsig. *Zen and the Art of Motorcycle Maintenance*. Bantam, 1984.

PJ88a. Pearl and J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.

PJ88b. Steven Pinker and JacquesMehler. *Connections and Symbols*. MIT Press, 1988.

PNR07. Karalny Patterson, Peter J. Nestor, and Timothy T. Rogers. Where do you know what you know? the representation of semantic knowledge in the human brain. *Nature Reviews Neuroscience*, 8:976–987, 2007.

PW78. D. Premack and G. Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and Brain Sciences*, pages 515–526, 1978.

QaGKKF05. R. Quian Quiroga, L. Reddy amd G. Kreiman, C. Koch, and I. Fried. Invariant visual representation by single-neurons in the human brain. *Nature*, 435:1102–1107, 2005.

QKKF08. R. Quian Quiroga, G Kreiman, C Koch, and I. Fried. Sparse but not "grandmother-cell" coding in the medial temporal lobe. *Trends in Cognitive Sciences*, 12:87–91, 2008.

RA98. G. Rizzolatti and M. A. Arbib. Language within our grasp. *Trends in Neurosciences*, 21:188–194, 1998.

Ram06. V.S. Ramachandran. Mirror neurons and imitation learning as the driving force behind 'the great leap forward' in human evolution. 2006. http://www.edge.org/3rd_culture/ramachandran/ramachandran_p1.html.

Rav04. Ian Ravenscroft. Folk psychology as a theory, stanford encyclopedia of philosophy. http://plato.stanford.edu/entries/folkpsych-theory/, 2004.

RBW92. Gagne R., L. Briggs, and W. Walter. *Principles of Instructional Design*. Harcourt Brace Jovanovich, 1992.

RC04. Giacamo Rizzolatti and Laila Craighero. The mirror-neuron system. *Annu. Rev. Neurosci.*, 27, 2004.

RCC93. D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. 1993.

RCK01. J. Rosbe, R. S. Chong, and D. E. Kieras. Modeling with perceptual and memory constraints: An epic-soar model of a simplified enroute air traffic control task. *SOAR Technology Inc. Report*, 2001.

Rie73. K. Riegel. Dialectic operations: the final phase of cognitive development. *Human Development*, 16.:346–370, 1973.

RM95. H. L. Roediger and K. B. McDermott. Creating false memories: Remembering words not presented in lists. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 21:803–814, 1995.

Ros88. Israel Rosenfield. *The Invention of Memory: A New View of the Brain*. Basic Books, 1988.

Ros99. J. Rosca. Genetic programming acquires solutions by combining top-down and bottom-up refinement. In *Foundations of Generic Programming*, 1999.

Row90. John Rowan. *Subpersonalities: The People Inside Us*. Routledge Press, 1990.

RV01. Alan Robinson and Andrei Voronkov. *Handbook of Automated Reasoning*. MIT Press, 2001.

RZDK05. Michael Rosenstein, ZvikaMarx, Tom Dietterich, and Leslie Pack Kaelbling. Transfer learning with an ensemble of background tasks. *NIPS workshop on inductive transfer*, 2005.

SA93. L. Shastri and V. Ajjanagadde. From simple associations to systematic reasoning: A connectionist encoding of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral & Brain Sciences*, 16-3, 1993.

Sal93. Stan Salthe. *Development and Evolution*. MIT Press, 1993.

SB67. G Spencer Brown. *Laws Of Form*. Cognizer, 1967.

SB98. Richard Sutton and Andrew Barto. *Reinforcement Learning*. MIT Press, 1998.

SBC05. S. Singh, A. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. *Proc. of Neural Information Processing Systems 17*, 2005.

SC94. Barry Smith and Roberto Casati. *Naive Physics: An Essay in Ontology*. Philosophical Psychology, 1994.

Sch91. Juergen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. *Proc. of the International Conf. on Simulation of Adaptive Behavior: From Animals to Animats*, 1991.

Sch06. Juergen Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 2006.

SDCCK08a. Steven Schockaert, Martine De Cock, Chris Cornelis, and Etienne E. Kerre. Fuzzy region connection calculus: An interpretation based on closeness. *Int. J. Approx. Reasoning*, 48(1):332–347, 2008.

SDCCK08b. Steven Schockaert, Martine De Cock, Chris Cornelis, and Etienne E. Kerre. Fuzzy region connection calculus: Representing vague topological information. *Int. J. Approx. Reasoning*, 48(1):314–331, 2008.

SE07. Stuart Shapiro and Al. Et. Metacognition in sneps. *AI Magazine*, 28, 2007.

Sha76. G. Shafer. *A Mathematical Theory of Evidence*. Princeton, NJ: Princeton University Press, 1976.

Shu03. Thomas R. Shultz. *Computational Developmental Psychology*. MIT Press, 2003.

Slo01. Aaron Sloman. Varieties of affect and the cogaff architecture schema. In *Proceedings of the Symposium on Emotion, Cognition, and Affective Computing*, AISB-01, 2001.

Slo08. Aaron Sloman. *The Well-Designed Young Mathematician*. Artificial Intelligence, December 2008.

SM09. R Sinha and R Mihalcea. Unsupervised graph-based word sense disambiguation. In N Nicolov and R Mitkov, editors, *Current Issues in Ling. Theory: Rec. Adv. in NLP*. John Benjamins, 2009.

SMI97. F-R. Sinot, Fernandez M., and Mackie I. Efficient reductions with director strings. *Evolutionary Computation*, 1997.

SMK12. Jeremy Stober, Risto Miikkulainen, and Benjamin Kuipers. Learning geometry from sensorimotor experience. In *Proceedings of the First Joint Conference on Development and Learning and Epigenetic Robotics*, 2012.

Sol64a. Ray Solomonoff. *A Formal Theory of Inductive Inference, Part I*. Information and Control, 1964.

Sol64b. Ray Solomonoff. *A Formal Theory of Inductive Inference, Part II*. Information and Control, 1964.

Sot11. Kaj Sotala. 14 objections against ai/friendly ai/the singularity answered. *Xuenay.net*, 2011. http://www.xuenay.net/objections.html , downloaded 3/20/11.

Spe96. L. Spector. Simultaneous evolution of programs and their control structures. In *Advances in Genetic Programming 2*. MIT Press, 1996.

SR04. Murray Shanahan and David A Randell. A logic-based formulation of active visual perception. In *Knowledge Representation*, 2004.

SS74. Jean Sauvy and Simonne Suavy. *The Child's Discovery of Space: From hopscotch to mazes – an introduction to intuitive topology*. Penguin, 1974.

SS03a. R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Lecture Notes in Computer Science vol. 2706*, 2003.

SS03b. John F. Santore and Stuart C. Shapiro. Crystal cassie: Use of a 3-d gaming environment for a cognitive agent. In *Papers of the IJCAI 2003 Workshop on Cognitive Modeling of Agents and Multi-Agent Interactions*, 2003.

ST93. Daniel Sleator and Davy Temperley. Parsing english with a link grammar. *Third International Workshop on Parsing Technologies.*, 1993.

Stc00. Theodore Stcherbatsky. *Buddhist Logic*. Motilal Banarsidass Pub, 2000.

SV99. A. J. Storkey and R. Valabregue. The basins of attraction of a new hopfield learning rule. *Neural Networks*, 12:869–876, 1999.

SW05. Reza Shadmehr and Steven P. Wise. *The Computational Neurobiology of Reaching and Pointing : A Foundation for Motor Learning*. MIT Press, 2005.

SZ04. R. Sun and X. Zhang. Top-down versus bottom-up learning in cognitive skill acquisition. *Cognitive Systems Research*, 5, 2004.

TC97. M. Tomasello and J. Call. *Primate Cognition*. Oxford University Press, 1997.

TC05. Endel Tulving and R. Craik. *The Oxford Handbook of Memory*. Oxford U. Press, 2005.

TM95. S. Thrun and Tom Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 1995.

Tom03. Michael Tomasello. *Constructing a Language: A Usage-Based Theory of Language Acquisition*. 2003.

TS94. E. Thelen and L. Smith. *A Dynamic Systems Approach to the Development of Cognition and Action*. MIT Press, 1994.

TS07. M. Taylor and P. Stone. Cross-domain transfer for reinforcement learning. *Proc. of the 24th International Conf. on Machine Learning*, 2007.

TSH11. Mohamad Tarifi, Meera Sitharam, and Jeffery Ho. Learning hierarchical sparse representations using iterative dictionary learning and dimension reduction. In *Proc. of BICA 2011*, 2011.

Tur77. Valentin F. Turchin. *The Phenomenon of Science*. Columbia University Press, 1977.

TV96. Turchin and V. Supercompilation: Techniques and results. In Dines Bjorner, M. Broy, and Aleksandr Vasilevich Zamulin, editors, *Perspectives of System Informatics*. Springer, 1996.

TVCC05. M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue. A study of fitness distance correlation as a difficulty measure in genetic programming. *Evolutionary Computation*, 2005.

Var79. Francisco Varela. *Principles of Biological Autonomy*. North-Holland, 1979.

VK94. T. Veale and M. T. Keane. *Metaphor and Memory and Meaning in Sapper: A Hybrid Model of Metaphor Interpretation*. Proceedings of the workshop on Hybrid Connectionist Systems of ECAI94, at the 11th European Conference on Artificial Intelligence, 1994.

VO07. Tony Veale and Diarmuid O'Donoghue. *Computation and Blending*. Cognitive Linguistics, 2007.

Vyg86. Lev Vygotsky. *Thought and Language*. MIT Press, 1986.

WA10. Wendell Wallach and Colin Atkins. *Moral Machines*. Oxford University Press, 2010.

Wah06. Wolfgang Wahlster. *SmartKom: Foundations of Multimodal Dialogue Systems*. Springer, 2006.

Wal01. Henrik Walter. *Neurophilosophy of Free Will*. MIT Press, 2001.

Wan95. P. Wang. *Non-Axiomatic Reasoning System*. PhD Thesis, Indiana University. Bloomington, 1995.

Wan06. Pei Wang. *Rigid Flexibility: The Logic of Intelligence*. Springer, 2006.

Was09. Mark Waser. Ethics for self-improving machines. In *AGI-09*, 2009. http://vimeo.com/3698890.

Wel90. H. Wellman. *The Child's Theory of Mind*. MIT Press, 1990.

WF05. Ian Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

WH06. J. Weng and W. S. Hwangi. From neural networks to the brain: Autonomous mental development. *IEEE Computational Intelligence Magazine*, 2006.

Who64. Benjamin Lee Whorf. *Language, Thought and Reality*. 1964.

WHZ+00. J. Weng, W. S. Hwang, Y. Zhang, C. Yang, and R. Smith. Developmental humanoids: Humanoids that develop skills automatically,. *Proc. the first IEEE-RAS International Conf. on Humanoid Robots*, 2000.

Wik11. Wikipedia. Open source governance. 2011. http://en.wikipedia.org/wiki/Open_source_governance.

Win72. T. Winograd. *Understanding Natural Language*. 1972.

Win95. Stephan Winter. Topological relations between discrete regions. In *Advances in Spatial Databases 4th International Symposium, SSD 95*, pages 310–327. Springer, 1995.

Win00. Stephan Winter. Uncertain topological relations between imprecise regions. *Journal of Geographical Information Science*, 14(5):411–430, 2000.

Wit07. David C. Witherington. *The Dynamic Systems Approach as Metatheory for Developmental Psychology, Human Development*. 50, 2007.

WKB05. Nico Van De Weghe, Bart Kuijpers, and Peter Bogaert. A qualitative trajectory calculus and the composition of its relations. In *Proc. of GeoS*, pages 60–76. Springer-Verlag, 2005.

Wol02. Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

Yan10. King-Yin Yan. A fuzzy-probabilistic calculus for vagueness. In *Unpublished manuscript*, 2010. Subm.

YGSS10. S Yi, T Glasmachers, T. Schaul, and J Schmidhuber. Frontier search. In *Proc. of the 3rd Conf. on AGI*. Atlantis Press, 2010.

YKL+04. Sanghoon Yeo, Jinwook Kim, Sung Hee Lee, Frank Chongwoo Park, Wooram Park, Junggon Kim, Changbeom Park, and Intaeck Yeo. A modular object-oriented framework for hierarchical multi-resolution robot simulation. *Robotica*, 22(2):141–154, 2004.

Yud04. Eliezer Yudkowsky. Coherent extrapolated volition. *Singularity Institute for AI*, 2004. http://singinst.org/upload/CEV.html.

Yud06. Eliezer Yudkowsky. What is friendly ai? *Singularity Institute for AI*, 2006. http://singinst.org/ourresearch/publications/what-is-friendly-ai.html.

Zad78. L. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

ZH10. Ruoyu Zou and Lawrence B. Holder. Frequent subgraph mining on a single large graph using sampling techniques. In *International Conference on Knowledge Discovery and Data Mining archive. Proceedings of the Eighth Workshop on Mining and Learning with Graphs. Washington DC*, pages 171–178, 2010.

ZLLY08. Xiaotong Zhang, Weiming Liu, Sanjiang Li, and Mingsheng Ying. Reasoning with cardinal directions: an efficient algorithm. In *AAAI'08: Proc. of the 23rd national conference on Artificial intelligence*, pages 387–392. AAAI Press, 2008.

ZM06. Song-Chun Zhu and David Mumford. A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Vision*, 2006.