Probabilistic Logic Based Reinforcement Learning of Simple Embodied Behaviors in a 3D Simulation World

Ari HELJAKKA (heljakka@iki.fi), Ben GOERTZEL, Welter SILVA, Cassio PENNACHIN, Andre' SENNA, Izabela GOERTZEL
Novamente LLC

Abstract. Logic-based AI is often thought of as being restricted to highly abstract domains such as theorem-proving and linguistic semantics. In the Novamente AGI architecture, however, probabilistic logic is used for a wider variety of purposes, including simple reinforcement learning of infantile behaviors, which are primarily concerned with perception and action rather than abstract cognition. This paper reports some simple experiments designed to validate the viability of this approach, via using the PLN probabilistic logic framework, implemented within the Novamente AGI architecture, to carry out reinforcement learning of simple embodied behaviors in a 3D simulation world (AGISim). The specific experiment focused upon involves teaching Novamente to play the game of "fetch" using reinforcement learning based on repeated partial rewards. Novamente is an integrative AGI architecture involving considerably more than just PLN; however, in this "fetch" experiment, the only cognitive process PLN is coupled with is simple perceptual pattern mining; other Novamente cognitive processes such as evolutionary learning and economic attention allocation are not utilized, so as to allow the study and demonstration of the power of PLN on its own.

1.Background and Motivation

The role of embodiment in AGI is somewhat philosophically controversial. Some claim that it is irrelevant: that, although human intelligence is closely tied to human embodiment, to extend this feature to AGI is unnecessary anthropomorphism. As alternatives to physical sensations and actions, suggested sources of knowledge for AGI include e.g. search engines like Google, and databases like Cyc [1]. Others, on the other hand, argue that embodiment is a necessary aspect of intelligence [2] – that mind is by nature embodied, and that disembodied software programs will never achieve true intelligence. Our view lies between these extremes. We view embodiment as an extremely convenient but not strictly necessary aspect of AGI systems.

One of the main advantages of embodiment for AGI is simply that humans are embodied, which implies that the study of embodied AGI's can benefit from our knowledge and intuition about embodied *human* intelligence. It is also clear that embodiment forces the deep integration of various aspects of intelligence – e.g. broad, shallow pattern recognition (needed for perception), procedure learning (needed for action), cognition, and attention allocation (needed for economical use of resources in a shifting environment). And, where learning human language is concerned, embodiment gives the opportunity for robust symbol grounding [3].

On the other hand, perhaps the biggest drawback of embodiment from the AGI developer's perspective is pragmatic in nature. Building, maintaining and using robots

requires considerable effort and cost, and requires a different sort of expertise than AGI software development does. This lead to the proposal of using simulated embodiments existing in simulated 3D worlds. While dealing with simulation worlds also involves overhead in terms of time, cost and expertise, it is much milder in all these regards than physical robotics. No simulation world running on currently affordable hardware will be able to provide a fully accurate simulation of the perceptual and motor-control challenges faced by physical android robots. However, we suggest that contemporary simulation worlds, appropriately utilized, can nonetheless permit effective simulation of many of the cognitive challenges that physical robots face – and can provide a more-than-adequate environment for experimentation with embodied AGI.

With this philosophy in mind, we have created a 3D simulation world called AGISim [4], and begun using it to teach an AI system to control a simulated humanoid, in the context of interactions with a similar human-controlled humanoid. Within this framework we are pursuing an AI-teaching program loosely guided by Piagetan developmental psychology. Our current focus is on infant-level cognition, including basic phenomena such as the understanding of the permanence of objects and agents – and, the primary topic of the current paper, simple games like fetch, tag and hide-and-seek. The next phase of teaching will focus on "theory of mind" – on encouraging the AI system to come to its own understanding of the intentions and beliefs and knowledge of other cognitive agents, based on its interactions with them in the simulated world. Most of this paper will focus on a single example – how the Novamente AI system learns to play "fetch" in the AGISim simulation world. However, this example gains most of its interest and importance from its role as a small part of a larger picture involving embodied artificial cognitive development.

1.1

AGISim and Novamente

AGISim is being developed as an open-source project[1], led by the first two authors, and is based on the CrystalSpace[2] 3D game engine, which can be configured to display realistic physics. It allows AI systems and humans to control android agents, and to experience the simulated world via multiple senses, as well as having the capability to chat with each other directly through text. A similar approach with CrystalSpace was earlier used by John Santore and Stuart Shapiro, in using the "Sneps" paraconsistent logic system to control an agent called Crystal Cassie [5].

It is intended that the experience of an AGI controlling an agent in AGISim should display the main qualitative properties of a human controlling her body in the physical world. The simulated world should support the integration of perception, action and cognition in a unified learning loop. And, it should support the integration of information from a number of different senses, all reporting different aspects of a common world. With these goals in mind, we have created the initial version of AGISim as a basic 3D simulation of the interior of a building, with simulations of sight, sound, smell and taste. An agent in AGISim has a certain amount of energy, and can move around and pick up objects and build things. While not an exact simulation of any specific physical robot, the android agent an AI controls in AGISim is designed to bear sufficient resemblance to a simple humanoid robot to enable the fairly straightforward porting of control routines learned in AGISim to a physical robot.

[1] sourceforge.net/projects/agisim

[2] crystal.sourceforge.net

Our work with AGISim to date has focused on controlling android agents in AGISim using the Novamente AI Engine (or NAIE; [6]; [7]), a comprehensive unique AI architecture that synthesizes perception, action, abstract cognition, linguistic capability, short and long term memory and other aspects of intelligence, in a manner inspired by complex systems science. Its design is based on a common mathematical foundation spanning all these aspects, which draws on probability theory and algorithmic information theory, among other areas. Unlike most contemporary AI projects, it is specifically oriented towards artificial *general* intelligence (AGI), rather than being restricted by design to one narrow domain or range of cognitive functions. The NAIE integrates aspects of prior AI projects and approaches, including symbolic, neural-network, evolutionary programming and reinforcement learning.

The existing codebase is being applied in bioinformatics, NLP and other domains.

To save space, some of the discussion in this paper will assume a basic familiarity with NAIE structures such as Atoms, Nodes, Links, ImplicationLinks and so forth, all of which are described in previous references and in other papers in this volume.

1.2 Cognitive Development in Simulated Androids

Jean Piaget, in his classic studies of developmental psychology [8] conceived of child development as falling into four stages, each roughly identified with an age group: infantile, preoperational, concrete operational, and formal. While Piaget's approach is out-of-date in some ways, recent researchers have still found it useful for structuring work in computational developmental psychology [9] ; we have modified the Piagetan approach somewhat for usage in our own work (see [10]). The basic Piagetan stages are as follows:

- Infantile: Imitation, repetition, association. Object permanence – infants learn that objects persist even when not being observed.
- Preoperational: Abstract mental representations. Word-object and image-object associations become systematic rather than occasional. Simple syntax.
- Concrete: Abstract logical thought applied to the physical world: conservation laws; more sophisticated classification; theory of mind – an understanding of the distinction between what I know and what others know. Classification becomes subtler.
- Formal: Abstract deductive reasoning contextually and pragmatically applied, the process of forming then testing hypotheses, etc.

We have carried out learning experiments involving the NAIE and AGISim, corresponding to aspects of Piaget's early stages of development.

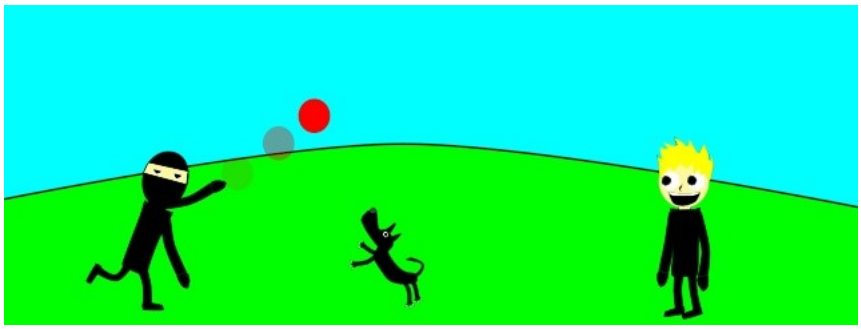
We have obtained results which suggest that a Novamente-powered simulated android can learn, via its interactions with human-controlled simulated android agents, to carry out basic cognitive tasks like word-object association and understanding the permanence of objects and agents. In particular, for object permanence, we have created a simulation of the "A-not-B" task commonly used in developmental psychology (see e.g. [11]), in which Novamente's ability to solve this task is specifically tied to its correct use of a specific inference rule called the "Rule of Choice." The teacher hides an object in location A repeatedly, then eventually hides it in location B and asks the AI agent to find it. Human babies less than 9 months of age

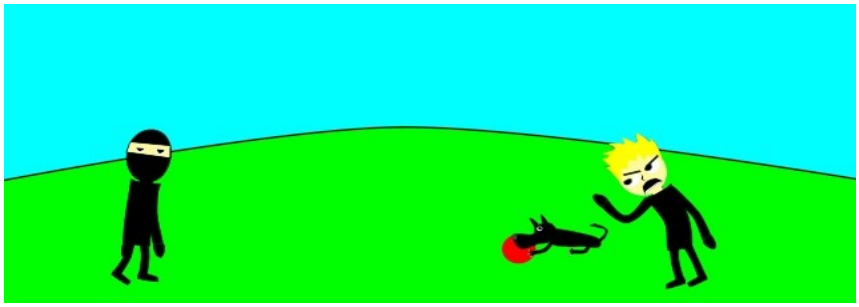
will often look in location A, but older babies look in B. The NAIE learns through interactive experience to look in location B – it learns that objects exist even when unobserved.

Another infantile example is the one that the bulk of this paper will focus on – "fetch", the game commonly played by dogs and their masters. Dogs and small children, as well as Novamente, easily master this game. However, the use of probabilistic logic in a dynamic environment where the inference premises must be formed from perceptions in real-time requires some fairly sophisticated probabilistic inference, as well as complex integration of other AI processes running in parallel. There are many other ways Novamente could learn to perform this task, e.g. using MOSES, or combining PLN and MOSES in various ways. It is not surprising that, given a simple task like fetch and a complex architecture like Novamente, there should be an oversupply of cognitive methods for solving the problem.

2. The Fetch Task

The basic idea underlying "fetch" is a simple one: the human throws an object and says "fetch," the dog runs to the object, picks it up, and brings it back to the human, who then rewards the dog for correct behavior. In our learning experiments, the teacher (a humanoid agent in AGISIm) plays the role of the human and the Novamente-controlled agent plays the role of the dog.





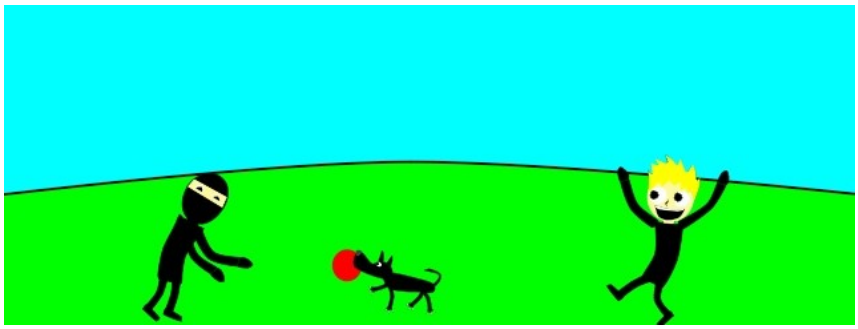


Figure 1. Dogs playing fetch, correctly (bottom) and incorrectly (middle). (Illustrations by Zebulon Goertzel.)

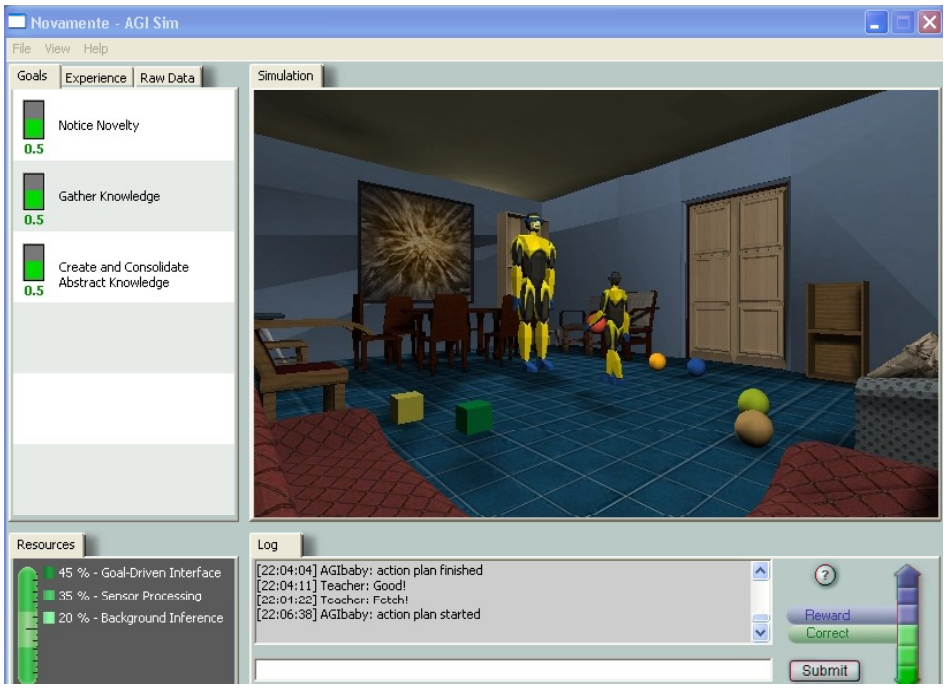


Figure 2: A screenshot of Novamente and its teacher playing fetch in the AGISim simulation world. Here Novamente is returning the ball to the teacher, after the teacher has thrown it.

1.

In more complex AGISim experiments, the teacher is actually controlled by a human being, who delivers rewards to Novamente by using the Reward controls on the AGISim user interface. In the fetch experiments reported here, because the task is so simple, the human controller was replaced by automated control code. The critical aspect of this automated teacher is partial reward. That is, you can't teach a dog (or a baby Novamente) to play fetch simply by rewarding it when it successfully retrieves the object and brings it to you, and rewarding it not at all otherwise – because the odds of it ever carrying out this "correct" behavior by random experimentation in the first place would be very low. What is needed for instruction to be successful is for there to be a structure of partial rewards in place.

We used here a modest approach with only one partial reward preceding the final reward for the target behaviour. The partial reward is given, first, whenever the agent manages to lift the thrown object. This reward is repeated several times, in order to make the agent learn that this behaviour implies reward, causing the agent to repeat this behaviour for some time even once the reward is removed, followed by some random actions. Finally, every time the rewarded behaviour happens to be followed by the agent carrying the object to the teacher and dropping it on her feet, the agent is rewarded again. In this manner, the agent learns the sequence of actions required for playing the game of "fetch." Given a more complex system of partial rewards, the agent can likewise be made to learn similar behaviour in a richer environment and with a larger set of possible actions.

3.

Learning Fetch within the Novamente Architecture

Novamente is an integrative AGI architecture, in which the highest levels of intelligence are intended to be achieved via the combination of several cognitive processes. However, fetch is a very simple task and does not require the full force of even the current incomplete version of the integrated Novamente cognition. From a Novamente point of view, it is interesting mostly as a "smoke test" for embodied reinforcement learning, to indicate that the basic mechanisms required for cognitive interaction with AGISim are integrated adequately and working correctly.

As noted above, fetch can be learned in Novamente in more than one way. The basic learning can be driven for example by PLN probabilistic reasoning or by MOSES evolutionary learning or by various combinations of aspects of the two. Here we will discuss only the PLN approach in depth.

3.1 Some Comments On Object Recognition

Firstly, in either the PLN or MOSES cases, learning fetch requires the assumption that the system already knows how to recognize objects: balls, teachers, and the like. I.e., the system must have learned (or been provided intrinsically with the capability for) "object recognition." We will make only a few comments about this here, as it is not the focus of this paper. Basically, object recognition, as the term is intended here, is the process of taking in raw data from the perceived environment (the AGISim world), and identifying in it collections of percepts that constitute "persistent objects." For instance, if the agent is looking at a scene consisting of a ball and a box, it needs to be able to identify that it is in fact looking at a ball and a box. And, if the ball rolls to the right slightly, it need to be able to identify that in some sense it is still looking at the same ball and the same box. This sort of capability develops fairly early in humans, but appears not to fully exist in very young babies. It seems very likely that its development in humans is a combination of learning with the progressive unfolding of genetically-programmed capabilities. Object recognition is not intrinsically dependent upon vision (see e.g. [12] for a treatment of the development of object recognition and related capabilities in blind babies), but in sighted people it is mainly dependent on vision, and the way we have handled it in AGISim so far is heavily vision-centric.

AGISim supports varying levels of visual granularity[1], including

- voxel vision: the system sees specific colors existing at specific coordinate points in the sim world.
- polygon vision: the system sees specific polygons with specific colors, and with corners at specific coordinate points in the sim world.
- object vision: the system sees specific polyhedral objects with corners at specific coordinate points.

The object-recognition problem is restricted to the first two cases. We have not dealt with the voxel vision case so far, but have decided that polygon vision represents a nice intermediary level: low enough to present the system with basic problems such as object recognition, yet high enough not to lead to the full computational complexity of human-style vision processing. Object recognition in the polygon vision case is a relatively simple matter of conjunctive pattern mining and clustering, but the specific algorithms associated with this process in Novamente will not be discussed here as they would lead us too far afield.

3.2 Inference, Pattern Mining and Predicate Schematization

Given object recognition, MOSES can learn to play fetch right away, without the need for any auxiliary learning processes. PLN could in principle do the same, with the help of an additional cognitive mechanism we call "predicate schematization." This process converts the logical knowledge obtained by PLN regarding how to play fetch into

concrete procedural knowledge that can be executed. Planning is a familiar manifestation of the general mechanism.

However, in order to make PLN learning of the fetch behavior reasonably efficient, an additional cognitive mechanism called "pattern mining" is also needed. This is a process that identifies frequent or otherwise significant patterns in a large body of data. The process is independent of whether the data is perceptual or related to actions, cognition etc. In principle, everything obtained via pattern mining could also be obtained via inference, but pattern mining has superior performance in many applications.

In either the PLN or MOSES approach, finally, the action of the Novamente component that actually executes learned procedures (called *schemata* in Novamente) is required.

So, in sum, the two minimal approaches to learning fetch in Novamente may be described as follows:

- object recognition + MOSES + schema execution
- object recognition + pattern mining + PLN + predicate schematization + schema execution

Neither of these approaches is optimal; for instance, greater learning efficiency may be obtained by integrating prioritization heuristics, which in Novamenre design are known as *economic attention allocation*, with MOSES and/or PLN so that the system wastes less time focusing on irrelevant things. But given the simplicity of the task, either of these approaches is sufficient. For this particular task, MOSES is a simpler but computationally slower approach than PLN; but that is really irrelevant, as our point here is not focused on the fetch task in itself (which obviously could be solved via much simpler methods than anything in Novamente) but on what it teaches us about the use of probabilistic inference in the context of embodied reinforcement learning.

4. Pattern Mining in Novamente

The "pattern mining" step mentioned above has not been discussed in previous publications on Novamente. We mention it briefly here. The pattern mining component is at the moment very simple, but may be made more sophisticated in future. The inputs of pattern mining are, in general,

- Atoms denoting raw outputs of the "sensors" that Novamente possesses in the AGISim world
- Atoms indicating actions Novamente has taken, e.g. in the AGISim world
- Potentially, other Atoms in Novamente's memory

The basic principle underlying pattern mining is to find *noteworthy combinations of inputs*. Furthermore, the mining process must find these via a purely "greedy search" methodology, without any slow and exploratory searching such as occurs for instance in evolutionary learning mechanisms like MOSES. Pattern mining has to operate in real time on potentially very large volumes of data (although in the current AGISim

configuration, the amount of data is in fact not very large due to the relatively small number and simpliciy of objects in the environment).

The current Novamente pattern miner operates via a simplistic "conjunction mining" approach. It is supplied with a set of standard *perceptual predicates*, which may be applied to its inputs. It then searches for conjunctions of these perceptual predicates, which occur surprisingly often in its experience.

For the fetch learning example, the perception process is relatively simple, and focused on the recognition of temporal patterns, such as those to be discussed in the following subsection.

4.1 Mining Temporal Patterns Relevant to Playing Fetch

All of the pattern mining relevant to the fetch learning example is temporal in nature. Frequent sequences of events must be recognized. As a simple example, sequences such as

```
SequentialAND
   SimultaneousAND
         I am holding the ball
         I am near the teacher
   I get more reward
```

Must be recognized. More formally, this looks like

```
SequentialAND
   SimultaneousAND
         holding(ball)
         near(me, teacher)
   Reward
```

Or in full-fledged Novamente Node/Link notation,

```
SequentialAND
   SimultaneousAND
         EvaluationLink holding ball
         EvaluationLink
               near
               ListLink (me, teacher)
   Reward
```

The predicates required here are near() and holding(), as well as the primitive "sensation" of Reward. Given these predicates as primitives, the mining of this conjunction from the system's experience is a simple matter. A similar approach works for mining conjunctions regarding other phases of the partial reward schedule used for fetch, as will be detailed below.

4.2 Intended Improvements to Novamente's Perceptual Pattern Mining Architecture

As a minor digression, it is anticipated that the simplistic approach to perceptual pattern mining discussed above may not hold up when we begin experimenting with a richer environment within AGISim (i.e. with environments containing a large number of complex objects); and so there is also a more complex design, not yet implemented, that involves embedding the conjunctive miner within a hierarchical architecture. Essentially, one may construct a hierarchical perception network (a subnetwork of Novamente's overall Atom network) in which each node refers to a certain localized region of spacetime, with the children of a node corresponding to the subregions of the region the node corresponds to. One may then carry out conjunctive mining within each local node of the hierarchical perception network.

Philosophically, this hierarchical perception approach displays significant similarities to Jeff Hawkins' [13] hierarchical perception architecture, though without his attempts at neurological justification. Hawkins' architecture relies on a combination of neural net activation spreading and Bayesian network based probabilistic calculations. On the other hand, in Novamente, we may spread attention between nodes in the perception network using economic attention allocation (similar to Hawkins' neural net activation spreading); and we may update the probabilities of conjunctions at various levels in the hierarchy using PLN probabilistic inference, which is ultimately just a different rearrangement of the mathematics used in Bayes nets (though PLN's arrangement of probability theory has significantly more general applicability). It may be noted that this combination of attentional and probabilistic dynamics also characterizes Maes behavior nets [14], which have some parallels to the action selection mechanisms in both Novamente and Stan Franklin's LIDA system [15]. It is with this kind of combination in mind that all Novamente Atoms have been supplied with both truth values and attention values.

5. PLN in the Fetch Example

As a prior chapter in this book [16] has already given a basic overview of PLN inference, this material will not be repeated here. Rather, a high-level overview of the application of PLN to the fetch problem will be given, together with a brief discussion of two relevant aspects of PLN that were not covered in the other chapter: inference about actions, and temporal inference.

From a PLN perspective, learning to play fetch is a simple instance of backward chaining inference. The goal of Novamente in this context is to maximize reward, and the goal of the PLN backward chainer is to find some way to prove that if some actionable predicates become true, then

```
Evaluation (Reward)
```

becomes true. This inference is possible by assuming that trying out actions is always possible, ie. the actions are considered to be in the axiom set of the inference. A more elegant approach we did not try yet would be to set a

```
PredictiveImplicationLink($1, Reward)
```

as the target of the inference, and launch the inference to fill in the variable slot $1 with a sequence of actions.

PredictiveImplicationLink is a Novamente Link type that combines logical (probabilistic) implication with temporal precedence. Basically, the backward chainer is being asked to construct an Atom that implies the future obtaining of reward. Each PredictiveImplicationLink contains a time-distribution indicating how long the target is supposed to occur after the source does; in this case the time-distribution must be centered around the rough length of time that a single episode of the "fetch" game occupies.

To learn how to play fetch, Novamente must repeatedly invoke PLN backward chaining on a knowledge base consisting of Atoms that are constantly being acted upon by perceptual pattern mining as discussed above. PLN learns logical knowledge regarding what circumstances imply reward, and then the predicate schematization process produces executable schemata embodying this knowledge, which are then executed – causing the system to carry out actions, which lead to new perceptions, which give PLN more information to guide its reasoning and lead to the construction of new procedures, etc.

The representation of temporal knowledge used in PLN (and Novamente generally) is based on a variant of the Event Calculus approach [17]. Here, we use a restricted set of predicates, only including the PredictiveImplicationLink and SequentialAnd(A B), which has the semantics "the event that A occurs, and then after A terminates, B initiates".

-
  -
  -
  -
  -

For instance in the case of

```
PredictiveImplicationLink A B
```

the truth value denotes (roughly speaking) the probability

```
P( event in class B initiates |
                event in class A terminates previously)
```

or more explicitly the average of w(x,y) over all (x,y) so that x is an event in A and y is an event in B, where w(x,y) is a "time distribution function" so that

- w(x,y)= 0 if x initiates after y terminates
- w(x,y) =1 if y's initiation is simultaneous with x's termination
- w(x,y) depends monotonically on the difference diff=(initiation of y – termination of x)

For example, in many cases one may use

```
w(x,y) = k/(diff+k)
```

where k is an adjustable parameter. For more details on temporal links and the event calculus variant we use, see [18].

In order to carry out very simple inferences about schema executions as required in the fetch example, two primitive predicates are used by PLN in Novamente:

- *try*, where try(X) indicating that the schema X is executed
- *can*, where can(X) indicates that the necessary preconditions of schema X are fulfilled, so that the execution of X will be possible

Furthermore, the following piece of knowledge is assumed to be known by the system, and is provided to Novamente as an axiom:

```
PredictiveImplication
   SimultaneousAnd
         Evaluation try X
         Evaluation can X
   Evaluation done X
```

That is: if the system can do X, and it tries to do X, then it has done X. Note that this implication may be used probabilistically, so it can be applied e.g. in cases where it is not certain whether or not the system can do X or not. If the system is unsure whether it can do X, then it will (according to this implication, without any additional knowledge) be unsure as to whether X will be done even if it tries to do X; and PLN's uncertain inference formulas may be used to estimate the latter uncertainty.

The proper use of the “can” predicate necessitates that we mine the history of occasions in which a certain action succeeded and occasions in which it did not. This allows us to create PredictiveImplications that embody the knowledge of the preconditions for successfully carrying out an action. In this paper we use a simpler approach because the basic mining problem is so easy: we just assume that “can” holds for all actions, and push the statistics of success/failure into the truth values of the PredictiveImplications produced by pattern mining. Hence, we don't try to determine the situation which enables the agent to carry out an action, but simply observe that in a certain percentage of cases the action succeeded, specifically in cases where it was part of a longer sequence of actions, so that the ealier actions likely contributed or fulfilled to the preconditions of the later action. We will soon illustrate this in the context of a real inference trail.

5.1 Inference Rules Used For Learning to Play Fetch

Finally, this section enumerates and explains the inference rules used in learning to play fetch.

Firstly, the ModusPonensRule is unsurprisingly a probabilistic version of modus ponens, i.e.

```
Implication A B
A
|-
B
```

Modus Ponens can also be applied to PredictiveImplications, insofar as the system keeps track of the structure of the proof tree so as to maintain the proper order of arguments. It is perhaps fortuitous that the order in which the arguments to Modus Ponens must be applied is always the same as the related temporal order, which allows us to extract a plan of consequtive actions, in an unambiguous order, from the proof tree.

Relatedly, the AndRule is of the form

```
A
B
|-
A & B
```

The AndRule can be accompanied with a temporal truth value formula so as to make them applicable for creating SequentialANDs. Later, we will only be concerned with SimpleANDRule, which is the AndRule that only looks at its constituents individually, and does not try to take advantage of partial conjunctions that could hold

useful information for the process of estimating the truth value of the complete conjunction.

The DeductionRule is of the form

```
Implication A B
Implication B C
|-
Implication A C
```

The PLN truth value formulas for these three rules are given in [18], in this volume.

The RewritingRule is a composition of AndRule and ModusPonensRule. It is used as a shorthand for converting atoms from one form to another when we have a Boolean true implication at our disposal. For these implications, the use of probabilistic inference is obviously unnecessary.

Finally, the function of the CrispUnificationRule is simply to produce, from a variable-laden universally quantified expression (atom), a version in which one or more variables has been bound. The truth value of the resulting atom is the same as that of the quantified expression itself.

6. Learning to Play Fetch via PLN Backward Chaining

This section describes one specific logical plan learned by PLN, based on the output of the perception miner, in order to play fetch in the AGISim world. The Novamente system is nondeterministic and different runs of the system, given the same environment and reward scheme, may lead to a variety of different internal plans and schemata. The learned plan discussed here is a representative one that lends itself relatively well to discussion.

Firstly, we define the specific predicates used as primitives for this learning experiment:

- Reward – a built-in sensation corresponding to the Novamente agent getting Reward, either via the AGISim teaching interface or otherwise via having its internal Reward indicator stimulated
- goto – a persistent event: goto(x) means the agent is going to x
- lift – an action: lift(x) means the agent lifts x
- drop – an action: drop(x) means the agent lifts x (and when this happens close to an agent T, we can interpret that informally as "giving" x to T)
- TeacherSay – a percept, TeacherSay(x) means that the teacher utters the string x

- holding – a persistent event, holding(x) means the agent is holding x
-

By assuming the above predicates, we are abstracting away from any actual motor learning: we are assuming that the system already knows how to lift and hold and drop, for example. Novamente's learning mechanisms appear to be capable of learning procedures to ground these motor actions, particularly in the simple context of the AGISim simulated robot, but these motor-learning experiments have not yet been done, so for the purpose of the fetch experiments reported here, we have simply assumed hard-coded procedures for these motor actions.

Also, note that we have assumed goto(x) as a built-in behavior. This is psychologically realistic, in the sense that before a dog or baby learns to play fetch, they have certainly already learned to move to an object that interests them. Learning goto(x) in the context of a very simple environment like the one used for these fetch experiments is very easy for Novamente, and unlike the motor learning experiments, this learning experiment has already been done. So the assumption made here is basically just that the fetch experiment must be done in a Novamente system that has already learned goto via prior teaching or spontaneous learning.

6.1 A Multi-Stage Partial Reward Function for Learning to Play Fetch

Next, we describe the partial reward function as a set of logical implications. The partial reward function is broken down into two stages, as described above. In these and following examples we use a shorthand notation, designed to improve legibility, e.g. PredImp for PredictiveImplicationLink, and "done goto ball" instead of

```
EvaluationLink
   done
   EvaluationLink goto ball
```

etc. The two stages of the reward function are:

Stage 1

```
PredImp
   holding ball
   Reward
```

Stage 2

```
PredImp
   SeqAnd
         holding ball
         done goto teacher
         done drop ball
   Reward
```

6.2

Knowledge Gained via Pattern Mining

Next, in the course of attempting to get rewarded by fulfilling the above partial reward functions, the pattern mining subsystem recognizes a number of conjunctions which PLN then evaluates and turns into implications. An example of the resulting implications is the following:

```
PredImp
   SeqAnd
         done goto Ball
         done lift Ball
   holding Ball
```

1.1.

6.3 *A PLN Inference Trajectory for Learning to Play Fetch*

Next, this bulk of this section shows a PLN inference trajectory which results in learning to play fetch according to the Stage 4 reward function mentioned above. This trajectory is one of many produced by PLN in various learning runs. When acted upon by the predicate schematization process, it produces the simple schema (executable procedure)

```
try goto Ball
try lift Ball
try goto Teacher
try drop Ball
```

It is quite striking to see how much work PLN and perception need to go through to get to this relatively simple plan! In fact, MOSES evolutionary learning can find it more simply, because the plan itself is quite small. However, MOSES occupies more runtime searching for the plan than PLN does, because MOSES essentially finds this plan through guided random search, rather than through understanding of the problem.

In a sense, MOSES has an easier time of it -- MOSES doesn't have to worry about the nature of "holding" at all, or the distinction between ongoing events and actions. However, in the process of learning this program, MOSES also does not build up as

much knowledge that is useful for solving other problems. PLN and perceptual pattern mining learn to play fetch by understanding each part of the "fetch" game and why it helps get partial reward, and then piecing together the behaviors corresponding to the different parts of the game into an overall plan. In the context of learning more complex tasks, of course MOSES and PLN may work together providing superior intelligence to what may be produced by either one separately. But in the context of this simple task of fetch, PLN can do the learning job quite efficiently with support only from pattern mining, without needing support from more sophisticated pattern recognition tools like MOSES.

The final inference trajectory follows.

First of all, the inference target was:

```
EvaluationLink (77) <0.80, 0.0099>  [4069]
  Reward:PredicateNode (26) <1, 0>  [191]
```

Note the truth value of the EvaluationLink initially found, which is <0.80, 0.0099>. Referring back to the definition of PLN truth values, this means that the inference process, after it had been running for suffiently long, found a way to achieve the Reward with a strength of 0.80, but with a weight of evidence of only .0099 (the rather unforgiving scaling factor of which originates from the internals of the perception miner). Continuing the run makes the strength increase towards, but not achieve, 1.0.

This target was produced by applying ModusPonensRule to the combination of

```
PredictiveImplicationLink <0.8,0.01> [9053948]
   SequentialAndLink <1,0> [9053937]
      EvaluationLink <1,0> [905394208]
         "holdingObject":PredicateNode <0,0> [6560272]
         "Ball":ConceptNode <0,0> [6582640]
      EvaluationLink <1,0> [905389520]
         "done":PredicateNode <0,0> [6606960]
         ExecutionLink [6888032]
               "goto":GroundedSchemaNode <0,0> [6553792]
               "Teacher":ConceptNode <0,0> [6554000]
      EvaluationLink <1,0> [905393440]
         "try":PredicateNode <0,0> [6552272]
         ExecutionLink [7505792]
               "drop":GroundedSchemaNode <0,0> [6564640]
               "Ball":ConceptNode <0,0> [6559856]
   EvaluationLink <1,0> [905391056]
      "Reward":PredicateNode <1,0> [191]
```

and

```
SequentialAndLink <1,0.01> [840895904]
   EvaluationLink <1,0.01> [104300720]
      "holdingObject":PredicateNode <0,0> [6560272]
      "Ball":ConceptNode <0,0> [6582640]
   EvaluationLink <1,1> [72895584]
      "done":PredicateNode <0,0> [6606960]
      ExecutionLink [6888032]
            "goto":GroundedSchemaNode <0,0> [6553792]
```

```
          "Teacher":ConceptNode <0,0> [6554000]
EvaluationLink <1,1> [104537344]
   "try":PredicateNode <0,0> [6552272]
   ExecutionLink [7505792]
         "drop":GroundedSchemaNode <0,0> [6564640]
         "Ball":ConceptNode <0,0> [6559856]
```

Graphically, the previous two link constructs would be denoted

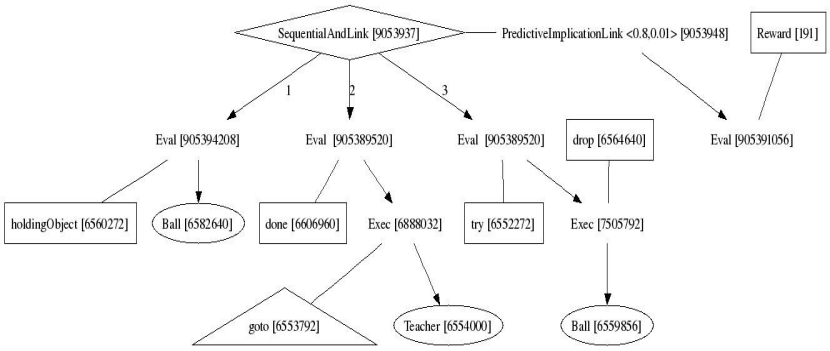


Figure 1.

and

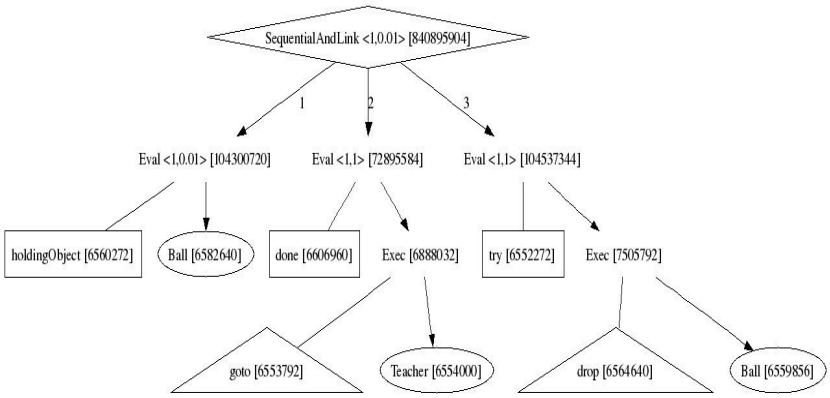


Figure 2.

Respectively. In the rest of this discussion we will often substitute graphical depictions for the indent-notation, in the interest of increasing comprehensibility.

Next, the SequentialANDLink [840895904] was produced by applying SimpleANDRule to its three child EvaluationLinks.

The EvaluationLink [104300720] was produced by applying ModusPonensRule to:

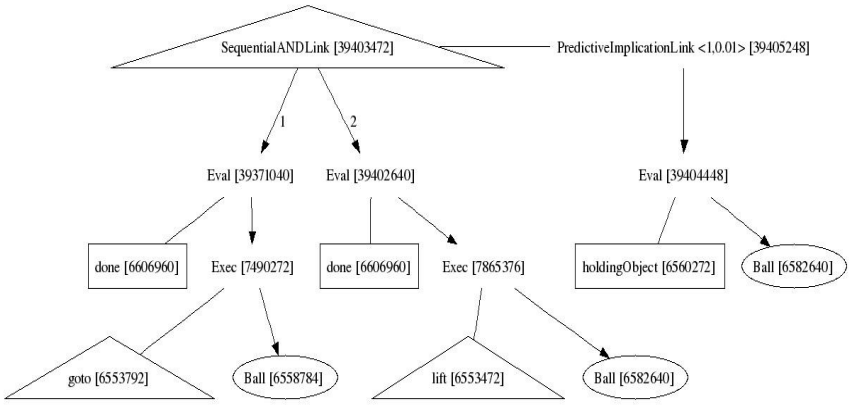


Figure 3.

which was mined from perception data, and to

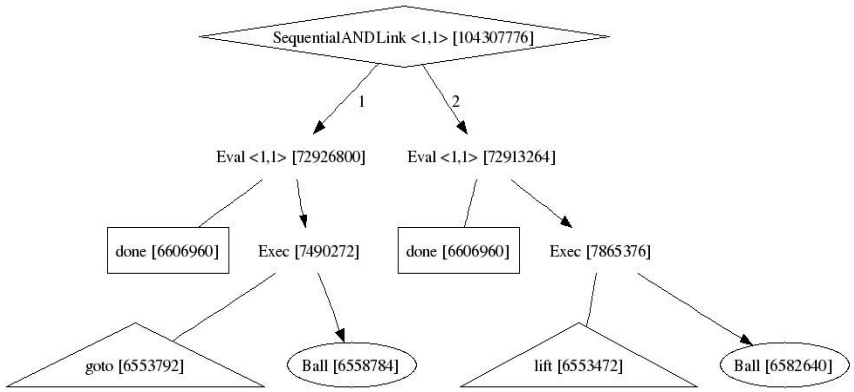


Figure 4.

The SequentialANDLink [104307776] was produced by applying SimpleANDRule to its two child EvaluationLinks. The EvaluationLink [72926800] was produced by applying RewritingRule to:

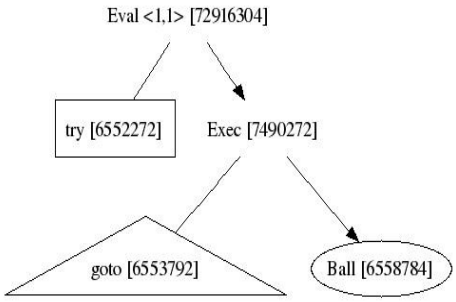


Figure 5.

and

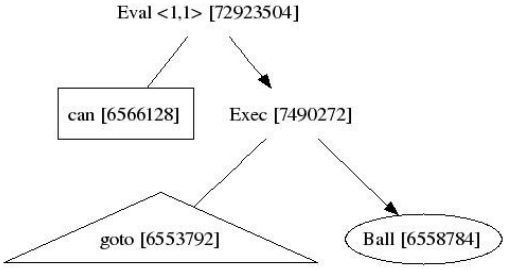


Figure 6.

The EvaluationLink [72916304], as well as all other *try* statements, were considered axiomatic, and technically produced by applying CrispUnificationRule to:

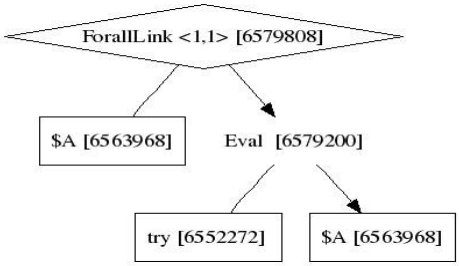


Figure 7.

The EvaluationLink [72923504], as well as all other *can* statements, were considered axiomatic, and technically produced by applying CrispUnificationRule to:

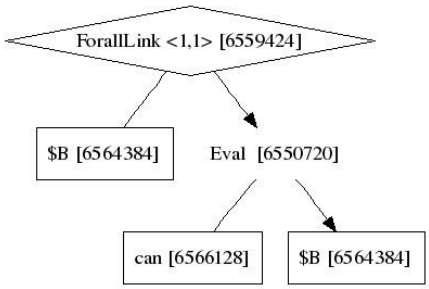


Figure 8.

The EvaluationLink [72913264] was produced by applying RewritingRule to:

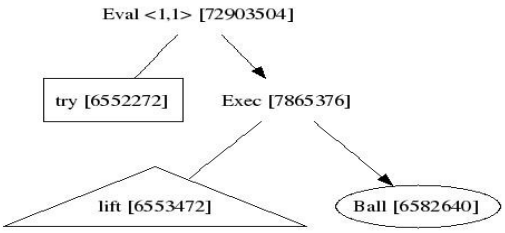


Figure 9.

and

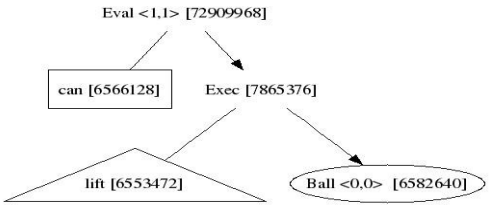


Figure 10.

Returning to the first PredictiveImplicationLink's children, EvaluationLink [72895584] was produced by applying RewritingRule to:

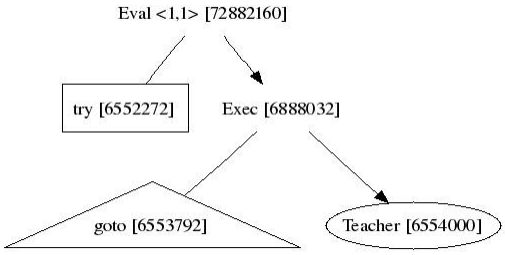


Figure 11.

and

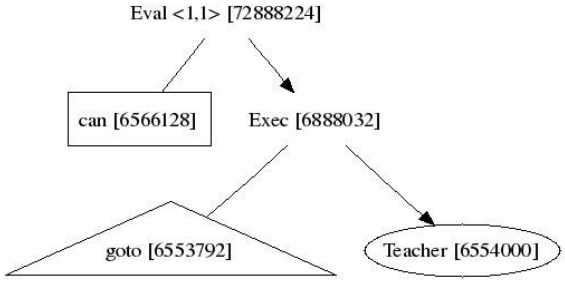


Figure 12.

which were both axiomatic.

QED!

For illustration, we finally present here an example of a plan the agent formed during the partial reward stage #1.

The inference target was:

```
EvaluationLink <0.83, 0.006>       [104296656]
  Reward:PredicateNode <1, 0>  [6565264]
```

[104296656] was produced by applying ModusPonensRule to:

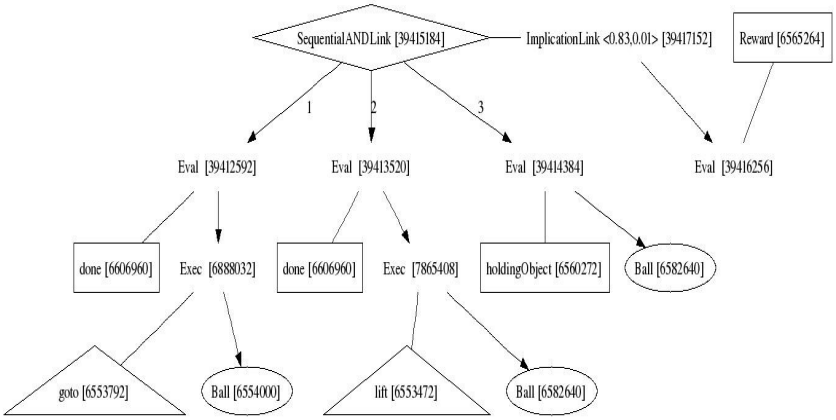


Figure 13.

On the other hand,

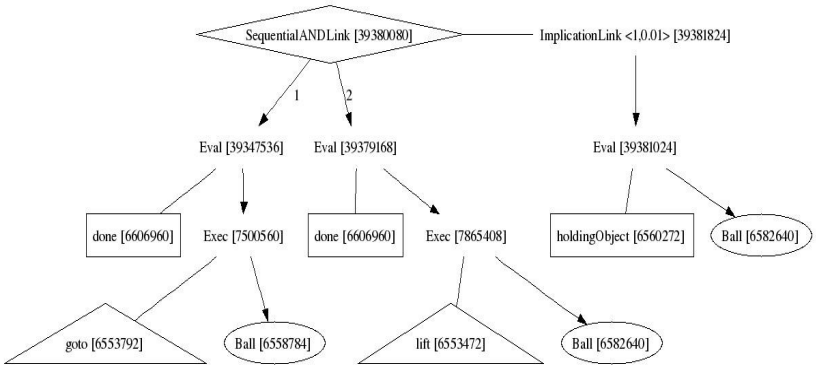


Figure 14.

This causes the system to come up with the following plan which works but contains obvious redundancy:

```
ExecutionLink <0,0.00062> [6888032]
```

```
     "goto":GroundedSchemaNode <0,0.00062> [6553792]
     "Ball":ConceptNode[typeId=10] <0,0.00062> [6554000]
ExecutionLink <0,0.00062> [7865408]
     "lift":GroundedSchemaNode <0,0.00062> [6553472]
     "Ball":ConceptNode <0,0.00062> [6582640]
ExecutionLink <0,0.00062> [7500560]
     "goto":GroundedSchemaNode <0,0.00062> [6553792]
     "Ball":ConceptNode[typeId=10] <0,0.00062> [6558784]
ExecutionLink <0,0.00062> [7865408]
     "lift":GroundedSchemaNode <0,0.00062> [6553472]
     "Ball":ConceptNode <0,0.00062> [6582640]
```

This redundancy may then be automatically removed by a simple reduction process. However, it is interesting that the said redundancy is not present in the previously described final plan, because the pattern miner was later able to find the more compact way to predict the occurrence of a Reward. So an explicit reduction step is not even necessary as the system eventually removes the redundancy on its own.

Summary and Conclusions

In this paper we have given a relatively detailed treatment of an extremely simple learning experiment – learning to play fetch -- conducted with the Novamente AGI system in the AGISim simulation world. In this conclusion we will discuss some of the general lessons we learned in the course of doing this work, particularly regarding the relationship between AGI and narrow AI; and we will reflect on some ways that this particular work reflects the general developmental strategy being taken within the Novamente AGI project.

Firstly, one interesting point to mention is the relatively high level of complexity and effort required to get the Novamente AGI architecture up to the point where learning to play fetch was possible. The vast majority of the ideas that went into the process of getting the Novamente system to learn to play fetch were omitted here due to space constraints! For instance, there are various subtleties related to the use of temporal knowledge within PLN backward chaining, to the execution of schemata pertaining to ongoing actions such as "holding" in AGISim, and so forth. The reason this is worth of mention is simply to illustrate the complexity required under the hood in order for learning of even very simple tasks to occur in an "AGI-friendly" way. Of course, it would be very simple to write a software program that "learned to play fetch," if the task were represented for the program in a sufficently direct way. The machine learning problem underlying fetch is an exceedingly simple one.

The essential point is that Novamente's learning to play fetch was not completely trivial because our approach was to first build an AGI architecture we believe to be capable of general learning, and only then apply it to the fetch test, while making minimal parameter adjustment to the specifics of the learning problem. This means that, in learning to play fetch, the system has to deal with perception, action and cognition modules that are not fetch-specific, but are rather intended to be powerful enough to deal with a wide variety of learning tasks corresponding to the full range of levels of cognitive development. Making all this "general infrastructure" work together to yield a simple behavior like fetch is a lot of work – the infrastructure doesn't increase the basic computational complexity of learning to play fetch, beyond what would be there in a simple fetch-specific learning system, but it adds a lot of "constant overhead."

Ultimately, in a problem this simple, the general-intelligence infrastructure doesn't add much. For instance, the PLN system is capable of powerful analogical reasoning, which means that once the system has learned to play fetch, it will be more easily able to learn to play other similar games afterwards. This capability will allow it to more easily carry out other tasks based on what it learned via learning to play fetch (a topic for a future paper). But in terms of the fetch task in isolation, PLN's capability for analogical inference doesn't help, and the fact of using a complex inference engine with so many capabilities merely complicates things. This paper marks merely the beginning of a series of publications involving more and more complex perception-cognition-action experiments we will undertake with this infrastructure.

Extending the scope of the discussion a little bit, these observations relate to some general differences between narrow AI and AGI work, which we believe are partly responsible for the relatively slow pace of the latter in the history of AI. For nearly any sufficiently narrowly-defined task, there is going to be some simplified, specialized approach that works reasonably well and is a lot easier to deal with than trying to apply a general-purpose AGI architecture to the problem. However, experience shows that taking specialized approaches to narrowly-defined problems yields minimal progress toward AGI. There are some problems, such as natural language conversation and autonomous scientific hypothesis, for which specialized approaches have proved almost totally ineffective – and work on making specialized systems to approach apparently-related tasks (e.g. text search, data mining) appears to help with these "AGI hard problems" hardly at all. Our hypothesis is that to approach these AGI-hard problems, the right approach is to construct an AGI architecture that in principle appears capable of solving these hard problems – and then, in order to test, tune and refine the AGI architecture, apply it to simpler problems related to the hard problems, *not worrying about the fact that the simpler problems may in fact be more easily solvable using narrowly specialized means*. That is the motivation for the work on fetch, object permanence, easter-egg hunting and other simple embodied-learning tasks currently being carried out with the Novamente AI Engine.

It is easy to draw an relevant analogy between learning in baby humans and baby animals. Puppies learn to play fetch more easily than human babies – probably, in part, because they are dealing with a less powerfully generalizable learning capability. The human baby has to tune a bigger, more general learning machine to the "fetch" task – but once it has done so, it is much better able to generalize this knowledge to other tasks, and build on it indirectly via inferential and other cognitive processes.

Acknowledgements

Thanks are due to those who helped with AGISim (including Sanjay Padmane, Teemu Keinonen, and others) and Novamente in its various aspects (including Moshe Looks, Thiago Maia and others), and earlier versions of PLN (mainly Guilherme Lamacie, the late Jeff Pressing, and Pei Wang).

Endnotes

References

[1] Guha, R. V. and Lenat, D. B. (1990). Cyc: A midterm report. AI Magazine, 11(3).

[2] Varelna, F., Thompson E. and Rosch, E. (1993). The Embodied Mind. The MIT Press

[3] Harnad, S. (1990) The Symbol Grounding Problem. Physica D 42:pp. 335-346

[4] Goertzel, Ben, Ari Heljakka, Stephan Vladimir Bugaj, Cassio Pennachin, Moshe Looks (2006). Exploring Android Developmental Psychology in a Simulation World, Symposium "Toward Social Mechanisms of Android Science", Proceedings of ICCS/CogSci 2006, Vancouver

[5] Santore, J. and Shapiro, S. C (2003). Crystal cassie: Use of a 3-d gaming environment for a cognitive agent. In Papers of the IJCAI 2003.

[6] Goertzel, B. (2006). A Probabilistic Event Calculus. (An unpublished technical report.) http://www.goertzel.org/new_research/ProbabilisticEventCalculus.pdf

[7] Goertzel, B., Looks, M., Heljakka, A. & Pennachin, C. (2006). "Toward a Pragmatic Understanding of the Cognitive Underpinnings of Symbol Grounding", Semiotics and Intelligent Systems Development, Ricardo Gudwin & João Queiroz, Eds., 2006

[8] Inhelder, B. and J. Piaget (1958). The Growth of Logical Thinking from Childhood to Adolescence. New York: Basic Books.

[9] Shultz, T. (2003). Computational Developmental Psychology. MIT Press.

[10] Goertzel, B. and Bugaj, S.V. (2006). Stages of Cognitive Development in Uncertain AI Systems, this volume

[11] Thelen and Smith (1994) A Dynamic Systems Approach to the Development of Cognition and Action, Cambridge, Mass.: MIT Press.

[12] Adelson, E. and Fraiberg, S. (1974), Gross motor development in infants blind from birth, Child Development, 45, 114-126

[13] Hawkins, J. and Blakeslee, S. (2004). On Intelligence. Times Books.

[14] Maes, P. (1991), A Bottom-up mechanism for Behavior Selection in an Artificial Creature, Proceedings of the first International Conference on Simulation of Adaptive Behavior, Meyer J.A. and Wilson S. (eds). MIT Press.

[15] Franklin, S. (2006). A Foundational Architecture for Artificial General Intelligence, this volume

[16] Ikle', M., Goertzel, B. and Goertzel, I. (2006). Quantifying Weight of Evidence in Uncertain Inference via Hybridizing Confidence Intervals and Imprecise Probabilities, this volume

[17] Miller, R. and Shanahan, M.(1999). The Event Calculus in Classical Logic - Alternative Axiomatisations. Electronic Transactions on Artificial Intelligence, Vol. 3 (1999), Section A, pp. 77-105.

[18] Ikle', M., Goertzel, B. and Goertzel, I. (2006). Quantifying Weight of Evidence in Uncertain Inference via Hybridizing Confidence Intervals and Imprecise Probabilities, this volume

[1] From a robotics vision perspective, it must be noted that even AGISim voxel vision is "cheating," in the sense that it involves the AI system directly receiving information about the distances of perceived voxels from its eyes. The human brain must infer distances using stereo vision, a complex matter in itself. However, some robots currently infer distances using lidar rather than stereopsis. In any case, in the Novamente project we have chosen not to get involved with low-level vision processing issues of this nature. When the time comes to interface Novamente with a physical robot, our approach will likely be to integrate an external vision-processing module that supplies either voxel or polygon vision inputs of the sort described above.