GOLEM: Toward an AGI Meta-Architecture Enabling Both Goal Preservation and Radical Self-Improvement

...

rough draft, circulated for preliminary commentary only

Ben Goertzel
Novamente LLC
1405 Bernerd Place
Rockville MD 20851

March 30, 2010

Abstract

A high-level AGI architecture called GOLEM (Goal-Oriented LEarning Meta-Architecture) is presented, along with an informal but careful argument that GOLEM may be capable of preserving its initial goals while radically improving its general intelligence. As a meta-architecture, GOLEM can be wrapped around a variety of different base-level AGI systems, and also has a role for a powerful narrow-AI subcomponent as a probability estimator. The motivation underlying these ideas is the desire to create AGI systems fulfilling the multiple criteria of being: massively and self-improvingly intelligent; probably beneficial; and almost surely not destructive.

1 Introduction

One question that looms large when thinking about the future of AGI is: How to create an AGI system that will maintain its initial goals even as it revises and improves itself – and becomes so much smarter that in many ways it becomes incomprehensible to its creators or its initial condition. One of the motives making this question interesting is the quest to design

AGI systems that are massively intelligent, probably beneficial, and almost surely not destructive.

Informally, I 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 is how to couple steadfastness with ongoing, radical, transformative learning.

I describe here an AGI meta-architecture, that I label the Goal Oriented LEarning Meta-architecture (GOLEM), and I present a careful but informal argument that, under certain reasonable assumptions, this architecture is likely to be both steadfast and massively, self-improvingly intelligent. Formalization of the argument is left for a later paper.

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 (e.g. some of my recent thinking about this has involved the notion of Coherent Aggregated Volition).

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:

- **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 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 Ss internal states and actions, as well as information about the environment during Ss 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
- **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 Testers 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)
  - 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 architecture as formulated above, it's worth noting that the GOLEM meta-architecture could be implemented using OpenCog or any other practical AGI architecture as a foundation – in this case, OpenCog is "merely" the initial condition for the OP, the Predictor and Searcher. However, demonstrating that self-improvement can proceed at a useful rate in any particular case like this, may be challenging.

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 [1].

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

[1]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

- 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.

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-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.

3 The Argument For GOLEM's Steadfastness

Our main goal here is to argue that a program with the 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.

4 Comparison to a Reinforcement Learning Based Formulation

Readers accustomed to reinforcement learning approaches to AI 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.

5 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 [2] AGI systems.

6 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 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

[2] A term used to refer to situations where a system rewires its reward or goal-satisfaction mechanisms to directly enable its own maximal satisfaction

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 toward 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.

7 Concluding Remarks

What we've sought to do here 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, and proofs (under appropriate assumptions) will be pursued for later publications.

It's clear that GOLEM can be wrapped around practical AGI architectures like OpenCog; 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 (OpenCog 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 OpenCog 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.