From Abstract Agents Models to Real-World AGI Architectures: Bridging the Gap

Ben Goertzel(✉)

OpenCog Foundation, Sha Tin, Hong Kong
ben@goertzel.org

Abstract. A series of formal models of intelligent agents is proposed, with increasing specificity and complexity: simple reinforcement learning agents; "cognit" agents with an abstract memory and processing model; hypergraph-based agents (in which "cognit" operations are carried out via hypergraphs); hypergraph agents with a rich language of nodes and hyperlinks (such as the OpenCog framework provides); "PGMC" agents whose rich hypergraphs are endowed with cognitive processes guided via Probabilistic Growth and Mining of Combinations; and finally variations of the PrimeAGI design, which is currently being built on top of the OpenCog framework.

1 Introduction

Researchers concerned with the abstract formal analysis of AGI have proposed and analyzed a number of highly simplified, mathematical models of generally intelligent agents (e.g. [11]). On the other hand, practical proto-AGI systems acting as agents in complex real-world situations, tend to have much more ad hoc, heterogenous architectures. There is no clear conceptual or mathematical bridge from the former world to the latter. However, such a bridge would have strong potential to provide guidance for future work from both the practical and formal directions.

To address this lack, we introduce here a hierarchy of formal models of intelligent agents, beginning with a very simple agent that has no structure apart from the requirement to issue actions and receive perceptions and rewards; and culminating with a specific AGI architecture, PrimeAGI[1] [9,10]. The steps along the path from the initial simple formal model toward OpenCog will each add more structure and specificity, restricting scope and making finer-grained analysis possible. Figure 1 illustrates the hierarchy to be explored.

The sequel paper [7] applies these ideas to provide a formal analysis of cognitive synergy, proposed as a key principle underlying AGI systems.[2]

[1] The architecture now labeled PrimeAGI was previously known as CogPrime, and is being implemented atop the OpenCog platform.

[2] The preprint [8] contains the present paper and the sequel, plus a bit of additional material.

T. Everitt et al. (Eds.): AGI 2017, LNAI 10414, pp. 3–12, 2017.
DOI: 10.1007/978-3-319-63703-7_1

2 Extending Basic Reinforcement Learning Agents

For the first step in our agent-model hierarchy, which we call a **Basic RL Agent** (RL for Reinforcement Learning), we will follow [11,12] and consider a model involving 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 in our model 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.

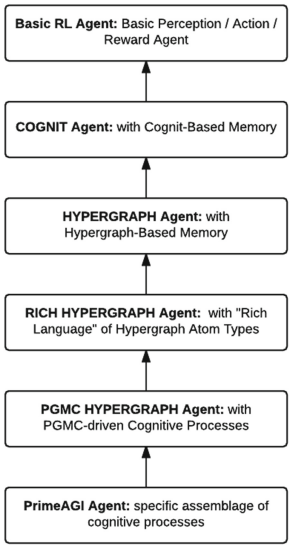

Fig. 1. An inheritance hierarchy showing the formal models of intelligent agents discussed here, with the most generic at the top and the most specific at the bottom.

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.

In [4] this formal agent model is extended in a few ways, intended to make it better reflect the realities of intelligent computational agents. First, the notion of a *goal* introduced, meaning a function that maps finite sequences $axs : t$ into rewards. As well as a distribution over environments, we have need for a conditional distribution γ, so that $\gamma(g, \mu)$ gives the weight of a goal g in the context of a particular environment μ. We assume that goals may be associated with symbols drawn from the alphabet $\mathcal{G}$. We also introduce a *goal-seeking agent*, which

is an agent that receives an additional kind of input besides the perceptions and rewards considered above: it receives goals.

Another modification is to allow agents to maintain memories (of finite size), and at each time step to carry out internal actions on their memories as well as external actions in the environment. Of course, this could in principle be accounted for within Legg and Hutter's framework by considering agent memories as part of the environment. However, this would seem an unnecessarily artificial formal model. Instead we introduce a set $\mathcal{C}$ of cognitive actions, and add these into the history of actions, observations and rewards.

Extending beyond the model given in [4], we introduce here a fixed set of "cognits" c_i (these are atomic cognitions, in the same way that the p_i in the model are atomic perceptions). Memory is understood to contain a mix of observations, actions, rewards, goals and cognitions. This extension is a significant one because we are going to model the interaction between atomic cognitions, and in this way model the actual decision-making, action-choosing actions inside the formal agent. This is big step beyond making a general formal model of an intelligent agent, toward making a formal model of a particular *kind* of intelligent agent. It seems to us currently that this sort of additional specificity is probably necessary in order to say anything useful about general intelligence under limited computational resources.

The convention we adopt is that: When a cognition is "activated", it acts – in principle – on all the other entities in the memory (though in most cases the result of this action on any particular entity may be null). The result of the action of cognition c_i on the entity x (which is in memory) may be any of:

- causing x to get removed from the memory ("forgotten")
- causing some new cognitive entity c_j to get created in (and then persist in) the memory
- if x is an action, causing x to get actually executed
- if x is a cognit, causing x to get activated

The process of a cognit acting on the memory may take time, during which various perceptions and actions may occur.

This sort of cognitive model may be conceived in algebraic terms; that is, we may consider $c_i * x = c_j$ as a product in a certain algebra. This kind of model has been discussed in detail in [3], where it was labeled a "self-generating system" and related to various other systems-theoretic models. One subtle question is whether one allows multiple copies of the same cognitive entity to exist in the memory. i.e. when a new c_j is created, what if c_j is already in the memory? Does nothing happen, or is the "count" of c_j in the memory increased? In the latter case, the memory becomes a multiset, and the product of cognit interactions becomes a (generally quite high dimensional, usually noncommutative and nonassociative) hypercomplex algebra over the nonnegative integers.

In this extended framework, an interaction/internal-action sequence may be written as

$$c_1 a_1 o_1 g_1 r_1 c_2 a_2 o_2 g_2 r_2 ...$$

with the understanding that any of the items in the series may be null. The meaning of c_i in the sequence is "cognit c_i is activated." One could also extend the model to explicitly incorporate concurrency, i.e.

$$c_{11}...c_{1k_{c1}}a_{11}...a_{1k_{a1}}o_{11}...o_{1k_{o1}}g_{11}...g_{1k_{g1}}$$
$$r_{11}...r_{1k_{r1}}c_{21}...c_{2k_{c2}}a_{21}...a_{2k_{a2}}o_{21}...o_{2k_{o2}}g_{21}...g_{2k_{g2}}r_{21}...r_{2k_{r2}}...$$

This **Cognit agent** is the next step up in our hierarchy of agents as shown in Fig. 1. The next step will be to make the model yet more concrete, by making a more specific assumption about the nature of the cognits being stored in the memory and activated.

3 Hypergraph Agents

Next we assume that the memory of our cognit-based memory has a more specific structure – that of a *labeled hypergraph*. This yield a basic model of a **Hypergraph Agent** – a specialization of the Cognit Agent model.

Recall that a hypergraph is a graph in which links may optionally connect more than two different nodes. Regarding labels: We will assume the nodes and links in the hypergraph may optionally be labeled with labels that are string, or structures of the form (string, vector of ints or floats). Here a string label may be interpreted as a node/link type indicator, and the numbers in the vector will potentially have different semantics based on the type.

Let us refer to the nodes and links of the memory hypergraph, collectively, as Atoms. In this case the cognits in the above formal model become either Atoms, or sets of Atoms (subhypergraphs of the overall memory hypergraph). When a cognit is activated, one or more of the following things happens, depending on the labels on the Atoms in the cognit:

1. the cognit produces some new cognit, which is determined based on its label and arity – and on the other cognits that it directly links to, or is directly linked to, within the hypergraph. Optionally, this new cognit may be activated.
2. the cognit activates one or more of the other cognits that it directly links to, or is directly linked to
   (a) one important example of this is: the cognit, when it is done acting, may optionally re-activate the cognit that activated it in the first place
3. the cognit is interpreted as a *pattern* (more on this below), which is then matched against the entire hypergraph; and the cognits returned from memory as "matches" are then inserted into memory
4. in some cases, other cognits may be removed from memory (based on their linkage to the cognit being activated)
5. nothing, i.e. not all cognits can be activated

Option 2*a* allows execution of "program graphs" embedded in the hypergraph. A cognit c_1 may pass activation to some cognit c_2 it is linked to, and then c_2 can do some computation and link the results of its computation to c_1, and then pass activation back to c_1, which can then do something with the results.

There are many ways to turn the above framework into a Turing-complete hypergraph-based program execution and memory framework. Indeed one can do this using only Option 1 in the above list. Much of our discussion here will be quite general and apply to any hypergraph-based agent control framework, including those that use only a few of the options listed above. However, we will pay most attention to the case where the cognits include some with fairly rich semantics.

The next agent model in our hierarchy is what we call an **Rich Hypergraph Agent**, meaning an agent with a memory hypergraph and a "rich language" of hypergraph Atom types. In this model, we assume we have Atom labels for "variable" and "lambda" and "implication" (labeled with a probability value) and "after" (with a time duration).; as well as for "and", "or" and "not", and a few other programmatic operators.

Given these constructs, we can use a hypergraph some of whose Atoms are labeled "variable" – such a hypergraph may be called an "h-pattern." We can also combine h-patterns using boolean operations, to get composite h-patterns. We can replicate probabilistic lambda calculus expressions explicitly in our hypergraph. And, given an h-pattern and another hypergraph H, we can ask whether P matches H, or whether P matches part of H.

To conveniently represent cognitive processes inside the hypergraph, it is convenient to include the following labels as primitives: "create Atom", "remove Atom", plus a few programmatic operations like arithmetic operations and combinators. In this case the program implementing a cognitive algorithm can be straightforwardly represented in the system hypergraph itself. (To avoid complexity, we can assume Atom immutability; i.e. make do only with Atom creation and removal, and carry out Atom modification via removal followed by creation.)

Finally, to get reflection, the state of the hypergraph at each point in time can also be considered as a hypergraph. Let us assume we have, in the rich language, labels for "time" and "atTime." We can then express, within the hypergraph itself, propositions of the form "At time 17:00 on 1/1/2017, this link existed" or "At time 12:35 on 1/1/2017, this link existed with this particular label". We can construct subhypergraphs expressing things like "If at time T an subhypergraph matching P exists, then s seconds after time T, a subhypergraph matching P_1 exists, with probability p."

The Rich Hypergraph and OpenCog. The "rich language" as outlined, is in essence a minimal version of the OpenCog AGI system[3]. OpenCog is based on a large memory hypergraph called the Atomspace, and it contains a number of cognitive processes implemented outside the Atomspace which act on the

[3] See http://opencog.org for current information, or [9,10] for theoretical background.

Atomspace, alongside cognitive processes implemented inside the Atomspace. It also contains a wide variety of Atom types beyond the ones listed above as part of the rich language. However, translating the full OpenCog hypergraph and cognitive-process machinery into the rich language would be straightforward if laborious.

The main reasons for not implementing OpenCog this way now are computational efficiency and developer convenience. However, future versions of OpenCog could potentially end up operating via compiling the full OpenCog hypergraph and cognitive-process model into some variation on the rich language as described here. This would have advantages where self-programming is concerned.

3.1 Some Useful Hypergraphs

The hypergraph memory we have been discussing is in effect a whole intelligent system – save the actual sensors and actuators – embodied in a hypergraph. Let us call this hypergraph "the system" under consideration (the intelligent system). We also will want to pay some attention to a larger hypergraph we may call the "meta-system", which is created with the same formalism as the system, but contains a lot more stuff. The meta-system records a plenitude of actual and hypothetical information about the system.

We can represent states of the system within the formalism of the system itself. In essence a "state" is a proposition of the form "h-pattern P_1 is present in the system" or "h-pattern P_1 matches the system as a whole." We can also represent probabilistic (or crisp) statements about transitions between system states within the formalism of the system, using lambdas and probabilistic implications. To be useful, the meta-system will need to contain a significant amount of Atoms referring to states of the system, and probabilistically labeled transitions between these states.

The implications representing transitions between two states, may be additionally linked to Atoms indicating the proximal cause of the transition. For the purpose of modeling cognitive synergy in a simple way, we are most concerned with the case in which there is a relatively small integer number of cognitive processes, whose action reasonably often cause changes in the system's state. (We may also assume some can occur for other reasons besides the activity of cognitive processes, e.g. inputs coming into the system, or simply random changes.)

So for instance if we have two cognitive processes called Reasoning and Blending, which act on the system, then these processes each correspond to a subgraph of the meta-system hypergraph: the subgraph containing the links indicating the state transitions effected by the process in question, and the nodes joined by these links. This representation makes sense whether or not the cognitive processes are implemented within the hypergraph, or a external processes acting on the system. We may call these "CPT graphs", short for "Cognitive Process Transition hypergraphs."

4 PGMC Agents: Intelligent Agents with Cognition Driven by Probabilistic History Mining

For understanding cognitive synergy thoroughly, it is useful to dig one level deeper and model the internals of cognitive processes in a way that is finer-grained and yet still abstract and broadly applicable.

4.1 Cognitive Processes and Homomorphism

In principle cognitive processes may be very diverse in their implementation as well as their conceptual logic. The rich language as outlined above enables implementation of anything that is computable. In practice, however, it seems that the cognitive processes of interest for human-like cognition may be summarized as sets of *hypergraph rewrite rules*, of the sort formalized in [1]. Roughly, a rule of that sort has an input h-pattern and an output h-pattern, along with optional auxiliary functions that determine the numerical weights associated with the Atoms in the output h-pattern, based on combination of the numerical weights in the input h-pattern.

Rules of this nature may be, but are not required to be, homomorphisms. One conjecture we make, however, is that for the cognitive processes of interest for human-like cognition, *most* of the rules involved (if one ignores the numerical-weights auxiliary functions) are in fact either hypergraph homomorphisms, or inverses of hypergraph homomorphisms. Recall that a graph (or hypergraph) homomorphism is a composition of elementary homomorphisms, each one of which merges two nodes into a new node, in a way that the new node inherits the connections of its parents. So the conjecture is

Conjecture 1. *Most operations undertaken by cognitive processes take the form either of:*

- *Merging two nodes into a new node, which inherits its parents' links*
- *Splitting a node into two nodes, so that the children's links taken together compose the (sole) parent's links*

(and then doing some weight-updating on the product).

4.2 Operations on Cognitive Process Transition Hypergraphs

One can place a natural Heyting algebra structure on the space of hypergraphs, using the disjoint union for $\sqcup$, the categorial (direct) product for $\sqcap$, and a special partial order called the cost-order, described in [6]. This Heyting algebra structure then allows one to assign probabilities to hypergraphs within a larger set of hypergraphs, e.g. to sub-hypergraphs within a larger hypergraph like the system or meta-system under consideration here. As reviewed in [6], this is an intuitionistic probability distribution lacking a double negation property, but this is not especially problematic.

It is worth concretely exemplifying what these Heyting algebra operators mean in the context of CPT graphs. Suppose we have two CPT graphs A and B, representing the state transitions corresponding to two different cognitive processes.

The meet $A \sqcap B$ is a graph representing transitions between conjuncted states of the system (e.g. "System has h-pattern P445 and h-pattern P7555", etc.). If A contains a transition between P_{445} and P_{33}, and B contains a transition between P_{7555} and P_{1234}; then, $A \sqcap B$ will contain a transition between $P_{445}\&P_{7555}$ and $P_{33}\&P_{1234}$. Clearly, if A and B are independent processes, then the probability of the meet of the two graphs will be the product of the probabilities of the graphs individually

The join $A \sqcup B$ is a graph representing, side by side, the two state transition graphs – as if we had a new process $A \text{or} B$, and a state of this new process could be either a state of A, or a state of B. If A and B are disjoint processes (with no overlapping states), then the probability of the join of the two graphs, is the sum of the probabilities of the graphs individually

The exponent A^B is a graph whose nodes are functions mapping states of B into states of A. So e.g. if B is a perception process and A is an action process, each node in A^B represents a function mapping perception-states into action-states. Two such functions F and G are linked only if, whenever node $b1$ and node $b2$ are linked in B, $F(b1)$ and $G(b2)$ are linked in G. I.e. F and G are linked only if $(F,G)(link(x,y)) = link(F(x),G(y))$, where by $(F,G)(link(x,y))$ one means the set $F(x), G(y)$.

So e.g. two perception-to-action mappings F and G are adjacent in action$^{\text{perception}}$ iff, whenever two perceptions p_1 and p_2 are adjacent, the action $a1 = F(p_1)$ is adjacent to the action $a2 = G(p_2)$. For instance, if

- F(perception p) = the action of carrying out perception p
- G(perception p) = the action done in reaction to seeing perception p

and

- p_1 = hearing the cat
- p_2 = looking at the cat

We then need

- $F(p_1)$ = the act of hearing the cat (cocking one?s ear etc.)
- $G(p_2)$ = the response to looking at the cat (raising ones eyes and making a startled expression)

to be adjacent in the graph of actions. If this is generally true for various (p_1, p_2) then F and G are adjacent in action$^{\text{perception}}$. Note that action$^{\text{perception}}$ is also the implication perception $\rightarrow$ action, where $\rightarrow$ is the Heyting algebra implication.

Finally, according to the definition of cost-based order $A < A_1$ if A and A_1 are homomorphic, and the shortest path to creating A_1 from irreducible source graph, is to first create A. In the context of CPT graphs, for instance, this

will hold if A_1 is a broader category of cognitive actions than A. If A denotes all facial expression actions, and A_1 denotes all physical actions, then we will have $A < A_1$.

4.3 PGMC: Cognitive Control with Pattern and Probability

Different cognitive processes may unfold according to quite different dynamics. However, from a general intelligence standpoint, we believe there is a common control logic that spans multiple cognitive processes – namely, adaptive control based on historically observed patterns. This process has been formalized and analyzed in a previous paper by the author [5], where it was called PGMC or "Probabilistic Growth and Mining of Combinations"; in this section we port that analysis to the context of the current formal model. This leads us to the next step in our hierarchy of agents models, a **PGMC Agent**, meaning an agent with a rich hypergraph memory, and homomorphism/history-mining based cognitive processes.

Consider the subgraph of a particular CPT graph that lies within the system at a specific point in time. The job of the cognitive control process (CCP) corresponding to a particular cognitive process, is to figure out what (if anything) that cognitive process should do next, to extend the current CPT graph. A cognitive process may have various specialized heuristics for carrying out this estimation, but the general approach we wish to consider here is one based on pattern mining from the system's history.

In accordance with our high-level formal agents model, we assume that the system has certain goals, which manifest themselves as a vector of fuzzy distributions over the states of the system. Representationally, we may assume a label "goal", and then assume that at any given time the system has n specific goals; and that, for each goal, each state may be associated with a number that indicates the degree to which it fulfills that goal.

It is quite possible that the system's dynamics may lead it to revise its own goals, to create new goals for itself, etc. However, that is not the process we wish to focus on here. For the moment we will assume there is a certain set of goals associated with the system; the point, then, is that a CCP's job is to figure out how to use the corresponding cognitive process to transition the system to states that will possess greater degrees of goal achievement.

Toward that end, the CCP may look at h-patterns in the subset of system history that is stored within the system itself. From these h-patterns, probabilistic calculations can be done to estimate the odds that a given action on the cognitive process's part, will yield a state manifesting a given amount of progress on goal achievement. In the case that a cognitive process chooses its actions stochastically, one can use the h-patterns inferred from the remembered parts of the system's history to inform a probability distribution over potential actions. Choosing cognitive actions based on the distribution implied by these h-patterns can be viewed a novel form of probabilistic programming, driven by fitness-based sampling rather than Monte Carlo sampling or optimization queries – this is the

"Probabilistic Growth and Mining of Combinations" (PGMC), process described and analyzed in [5].

Based on inference from h-patterns mined from history, a CCP can then create probabilistically weighted links from Atoms representing h-patterns in the system's current state, to Atoms representing h-patterns in potential future states. A CCP can also, optionally, create probabilistically weighted links from Atoms representing potential future state h-patterns (or present state h-patterns) to goals. It will often be valuable for these various links to be weighted with confidence values alongside probability values; or (almost) equivalently with interval (imprecise) probability values [2].

5 Conclusion

And so we have reconstructed the core concepts of the OpenCog platform and PrimeAGI architecture, via building up step by step from a simple reinforcement learning agent. One could proceed similarly for other complex cognitive architectures. The hope is that this sort of connection can help guide the extension of formal analyses of AGI in the direction of practical system architecture.

References

1. Baget, J.F., Mugnier, M.L.: Extensions of simple conceptual graphs: the complexity of rules and constraints. J. Artif. Intell. Res. **16**, 425–465 (2002)
2. Goertzel, B., Ikle, M., Goertzel, I., Heljakka, A.: Probabilistic Logic Networks. Springer, Heidelberg (2008)
3. Goertzel, B.: Chaotic Logic. Plenum, New York (1994)
4. Goertzel, B.: Toward a formal definition of real-world general intelligence. In: Proceedings of AGI 2010 (2010)
5. Goertzel, B.: Probabilistic growth and mining of combinations: a unifying meta-algorithm for practical general intelligence. In: Steunebrink, B., Wang, P., Goertzel, B. (eds.) AGI -2016. LNCS, vol. 9782, pp. 344–353. Springer, Cham (2016). doi:10.1007/978-3-319-41649-6_35
6. Goertzel, B.: Cost-based intuitionist probabilities on spaces of graphs, hypergraphs and theorems (2017)
7. Goertzel, B.: Toward a formal model of cognitive synergy. In: Proceedings of AGI 2017. Springer, Cham (2017, submitted)
8. Goertzel, B.: Toward a formal model of cognitive synergy (2017). https://arxiv.org/abs/1703.04361
9. Goertzel, B., Pennachin, C., Geisweiller, N.: Engineering General Intelligence, Part 1: A Path to Advanced AGI via Embodied Learning and Cognitive Synergy. Atlantis Thinking Machines, New York (2013). Springer
10. Goertzel, B., Pennachin, C., Geisweiller, N.: Engineering General Intelligence, Part 2: The CogPrime Architecture for Integrative, Embodied AGI. Atlantis Thinking Machines, New York (2013). Springer
11. Hutter, M.: Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability. Springer, Heidelberg (2005)
12. Legg, S.: Machine super intelligence. Ph.D. thesis, University of Lugano (2008)