Rational OpenCog Controlled Agent

Nil Geisweiller(✉) and Hedra Yusuf(✉)

SingularityNET Foundation, Amsterdam, The Netherlands
{nil,hedra}@singularitynet.io

Abstract. In this paper we introduce, ROCCA for *Rational OpenCog Controlled Agent*, an agent, that, as its name suggests, leverages the OpenCog framework to fulfill goals in uncertain environments. It attempts to act rationally, relying on reasoning for both learning and planning. An experiment in a Minecraft environment is provided as a test case.

Keywords: Symbolic Reinforcement Learning · Pattern Mining · Procedural Reasoning · Thompson Sampling · OpenCog · Minecraft

1 Introduction

This paper describes an attempt to leverage the OpenCog framework [15] for controlling agents in uncertain environments. It can be seen as a reboot of previous attempts [5,10,12] relying on new or improved components such as

- a hypergraph pattern miner [7] and a version of Probabilistic Logic Networks (PLN) [9] both implemented on top of OpenCog's Unified Rule Engine equipped with an inference control mechanism;
- a temporal and procedural extension of PLN [8];
- a simplified version of OpenPsi [5] leaving aside built-in urges and modulators from MicroPsi [3] and using an action selection policy based on Thompson Sampling [17].

It is comparable to OpenNARS for Applications (ONA) [14] but, among other differences, uses PLN [9] as its core logic.

The ultimate goal of this project is to provide a technology to enable us to experiment with forms of meta-learning and introspective reasoning for self-improvements. The rational for using a reasoning-based system is that it offers maximum transparency and is thus more amenable to reflectivity and introspection [11,19]. The work that is described in this paper is only the premise of that goal. No meta-learning is taking place yet. The objective for now is to build an agent that is able to discover regularities from its environment and acts rationally, possibly at the expense of efficiency, at least initially. For discovering regularities, the agent uses a reasoning-based pattern miner [7]. Then combine

P. Hammer et al. (Eds.): AGI 2023, LNAI 13921, pp. 95–104, 2023.
https://doi.org/10.1007/978-3-031-33469-6_10

these regularities to form plans using temporal and procedural reasoning. More specifically plans are predictive implications of the form

$$\boldsymbol{C} \wedge \boldsymbol{A} \rightsquigarrow^{\boldsymbol{T}} \boldsymbol{G} \stackrel{\text{m}}{=} TV$$

called *Cognitive Schematics* or *Schemata*. Which can be read as *"in some context* $\boldsymbol{C}$*, if some, possibly composite, action* $\boldsymbol{A}$ *is executed, then after* $\boldsymbol{T}$ *time units, the goal* $\boldsymbol{G}$ *is likely to be fulfilled, with second order probability measured by* TV*"*. These plans are then combined to into a mixture that grossly approximates Solomonoff distribution [6]. Finally, the next action is selected using Thompson Sampling [17]. The resulting system is called ROCCA for *Rational OpenCog Controlled Agent*.

The rest of the paper is organized as follows. A recall of the OpenCog framework is provided in Sect. 2. ROCCA is described in Sect. 3. An experiment using it to control an agent in Minecraft is described in Sect. 4. A conclusion including future directions is given in Sect. 5.

2 OpenCog Framework Recall

OpenCog [15] is a framework offering a hypergraph database technology with a query language and a collection of programs built on top of it to perform cognitive functions such as learning, reasoning, spreading attention and more. Knowledge is stored in *AtomSpaces*, hypergraphs composed of *atoms*, *links* and *nodes*, where links can connect to other atoms. *Values* can be attached to atoms to hold probability, confidence, importance and more. Values and atoms can be of various types. Let us recall the types we need for the rest of paper.

- A *TruthValue* is a second order probability distribution, i.e. a probability of a probability.
- A *SimpleTruthValue* is a TruthValue where the second order distribution is represented by a beta distribution of parameters controlled by a *strength*, a proxy for a probability, and a *confidence* over that strength. It is denoted $< s, c >$ where s is the strength and c is the confidence.
- A *Predicate* is function from a certain domain to *Boolean*. A TruthValue can be attached to a predicate, representing the prevalence of its satisfied inputs. For instance $\boldsymbol{P} \stackrel{\text{m}}{=} < 0.4, 0.1 >$ represents that $\boldsymbol{P}$ tends to evaluate to *True* 40% of the time, but there is a small confidence of 0.1 over that 40%. A TruthValue can be attached to individual evaluations as well. For instance $\boldsymbol{P}(a) \stackrel{\text{m}}{=} < 0.9, 1 >$ represents that the probability of $\boldsymbol{P}(a)$ evaluating over a particular a to *True*, is 0.9 and we are certain about it.
- A *Conjunction* is a link between two predicates, representing the predicate resulting from the pointwise conjunction of these two predicates. For instance $\boldsymbol{P} \wedge \boldsymbol{Q} \stackrel{\text{m}}{=} < 0.2, 0.3 >$ represents the prevalence, with strength 0.2 and confidence 0.3, of the pointwise conjunction of $\boldsymbol{P}$ and $\boldsymbol{Q}$.
- An *Implication* is a link between two predicates, semantically representing the conditional probability between two events represented by these predicates. For instance $\boldsymbol{P} \rightarrow \boldsymbol{Q} \stackrel{\text{m}}{=} < 0.7, 0.4 >$ indicates that if $\boldsymbol{P}(x)$ is *True* then there is a 70% change with a 0.4 confidence, that $\boldsymbol{Q}(x)$ is also *True*.

Additionally we use the following types for temporal reasoning.

- A *Sequential Conjunction* is a link between two temporal predicates, representing the predicate resulting from the pointwise conjunction of these predicates while the second one leads by a certain time. For instance $P \nearrow^T Q$ is the pointwise conjunction of P and a leading Q by T time units. Meaning that $(P \nearrow^T Q)(x,t)$ is *True* if and only if $P(x,t)$ and $Q(x,t+T)$ are *True*.
- A *Predictive Implication* is a link between two temporal predicates, representing the conditional probability between two events delayed by a certain time. For instance $P \rightsquigarrow^T Q \stackrel{m}{=} <0.8, 0.5>$ indicates that if $P(x)$ is *True* then there is a 80% chance with a 0.5 confidence, that after T time units $Q(x)$ will also be *True*.

The difference between a temporal and an atemporal predicate is its domain. A temporal predicate must have at least a temporal dimension. More detail about the temporal types and their associated inference rules is provided in [8].

3 Rational OpenCog Controlled Agent

ROCCA is implemented as an observation-planning-action loop interleaved with learning and reasoning. It provides an interfacing between OpenCog and environments such as Malmo [16] or OpenAI Gym [4]. It is written in Python which is both supported by these environments and OpenCog. Figure 1 provide a graphical representation of ROCCA as if it was a single loop incorporating all steps.

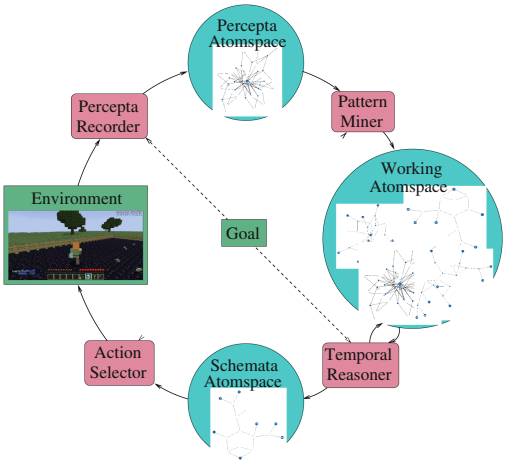

Fig. 1. Rational OpenCog Controlled Agent control and learning cycles merged into a single loop.

3.1 Memory

For better efficiency and clarity, the memory of the agent is split into three parts.

1. The *Percepta* AtomSpace holds timestamped observations as they come into the system.
2. The *Working* AtomSpace holds any kind of data, ranging from timestamped observations to predictive implications. Most knowledge used and inferred during the course of learning are usually dumped into this AtomSpace.
3. The *Schemata* AtomSpace holds *Cognitive Schematics*, which are predictive implications relating contexts, actions and goals.

3.2 Processes

The agent runs two main processes:

1. A *Control* process for real-time reactive agent control.
2. A *Learning* process for non-reactive background learning.

In principle these two processes could happen in parallel. For now they alternate in series. The agent starts in a control phase. A number of control cycles occur as the agent motor-babbles through its environment. It is then followed by a learning phase when the agent discover regularities and build plans. And finally repeats the control phase to test how it performs after learning.

3.3 Control

The control process is composed of control cycles, each decomposed into *Observation*, *Planning* and *Action* phases, as described below.

1. *Observation*:
   (a) Receive and timestamp observations, reward included, from the environment.
   (b) Store the timestamped observations in the Percepta AtomSpace.
2. *Planning*:
   (a) Select the goal for that cycle.
   (b) Find plans fulfilling that goal from the Schemata AtomSpace.
   (c) Build a mixture distribution from these plans.
3. *Action*:
   (a) Select the next action via Thompson Sampling according to that mixture distribution.
   (b) Timestamp and store the selected action in the Percepta AtomSpace.
   (c) Run the selected action and by that update the environment.

None of these steps are computationally expensive. They involve algorithms that are at most linear with the size of the Percepta and Schemata AtomSpaces. As time goes and knowledge accumulates though, it will progressively slow down. Indeed, for real-time responsiveness such control cycle should be bound by a constant. Achieving this may require to incorporate other mechanisms such as filtering and forgetting. This problem, as important as it is, is left aside for future research. Given the small environments ROCCA has been tested with, it has not been a problem so far. Let us now provide more details about these three phases.

Observation. During the observation phase, data coming from the environment are timestamped and stored in the Percepta AtomSpace with the format *datum@timestamp*. For instance if at cycle 3 the agent observes *outside(house)*, then *outside(house)@3* is inserted into the Percepta AtomSpace.

Planning. The first step of the planning phase is to select a goal G to fulfill. In the current version of ROCCA though it merely returns a constant goal which is to gain a reward within a forward window. More on goal selection can be found in [11,13]. Once the goal has been selected, the agent searches the Schemata AtomSpace with the following pattern matcher query

$$\$C \wedge \$A \rightsquigarrow^{T} G$$

where $\$C$ is a variable representing the context, $\$A$ is a variable representing the action, T is a time delay selected within that forward window and G is the selected goal. All returned candidates are then filtered according to their contexts, only retaining those for which the context is evaluated to *True* at the current time. Ideally, such crisp evaluation should be replaced by a second order probability evaluation of a context being *True*. This is important for contexts that have elements of uncertainty. But for the sake of simplicity, in our experiments so far, all contexts are crisply evaluated. Then from the set of valid cognitive schematics, a second order mixture distribution is built and handed to the next phase for performing action selection. The calculations used to build that second order mixture distribution is detailed in [6].

Action. The Action phase consists of the following steps:

1. Select the next action via Thompson Sampling [17] according to the mixture distribution built during the planning phase.
2. Timestamp and store the selected action in the Percepta AtomSpace.
3. Run the selected action and update the environment. If it is a composite action, only run the first primary action.

The trickiest step here is selecting the next action via Thompson Sampling. The novelty is that the second order probabilities can be leveraged by Thompson Sampling. For example, assume we have two actions, A_1 and A_2, to choose among two predictive implications

$$C_1 \wedge A_1 \rightsquigarrow^{T} G \stackrel{\text{m}}{=} <0.6, 0.9>$$
$$C_2 \wedge A_2 \rightsquigarrow^{T} G \stackrel{\text{m}}{=} <0.7, 0.1>$$

Using only the strengths of the truth values as proxy for probability of success, the choice is clear. Action A_2 should be selected, because its probability of success, which is 0.7, is greater than that of A_1, which is 0.6. However once confidence is introduced, that choice becomes less clear because the truth value of success of A_2 has a low confidence of 0.1. In that case, first order probabilities are sampled from their corresponding second order distributions, and then these probabilities are compared. The action with the maximum probability gets selected. Informally, the idea is to consider the possibilities that the agent might be living in a world where A_2 has a lower probability of success than A_1. That is the essence of Thompson Sampling. Figure 2 shows the second order distributions of the probabilities of success of A_1, in blue, and A_2, in red, for these

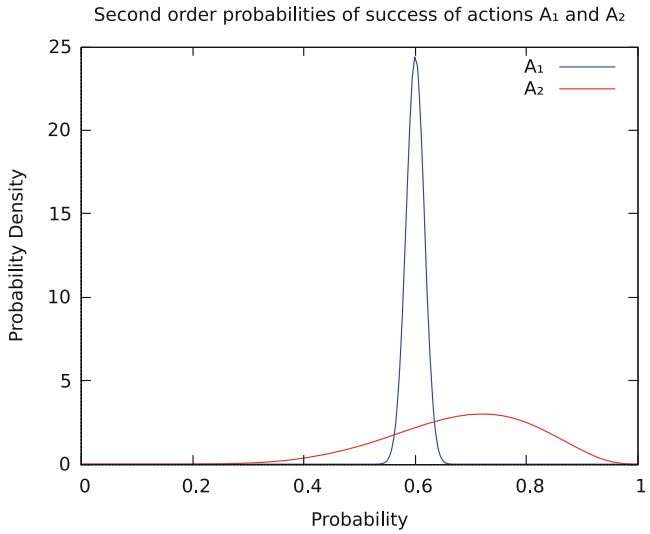

Fig. 2. Second order probability distributions of success of actions A_1 and A_2, using as parameters of the beta distribution $\alpha(s,c) = \alpha_0 + \frac{s.c.k}{1-c}$ and $\beta(s,c) = \beta_0 + \frac{(1-s).c.k}{1-c}$ where k, the *lookahead*, is set to 100, and α_0 and β_0 are set to Jeffreys prior.

truth values. As one may notice, the area under the red curve situated at the left of the blue curve is non-negligible. Meaning that the probability of being in a world where A_1 has a higher probability of success than A_2 is non-negligible as well. Because these strengths and confidences are periodically updated during the lifetime of the agent, one can see how Thompson Sampling is a great alternative to ε-greedy, as it offers a parameter-free mechanism to balance exploration and exploitation that dynamically adapts with the knowledge of the agent.

Note that in this example only two actions among two cognitive schematics are considered, but in practice there is usually a handful of actions among a potentially very large number of cognitive schematics with overlapping contexts and conflicting goals. The resulting distribution of success of each action is typically multi-modal and do not reduce to a beta distribution. How to deal with such a multitude of cognitive schematics is treated in [6].

3.4 Learning

The difficulty then comes down to discovering cognitive schematics that are as predictive and widely applicable as possible. For that, ROCCA uses a combination of pattern mining and reasoning.

Pattern Mining. A relatively inexpensive way to discover regularities in the environment is to mine the Percepta AtomSpace. For instance, given

$$\{go(right)@0,\ square(right)@1,\ go(left)@1,\ square(left)@2, \dots\}$$

the pattern miner can discover temporal relationships such as

$$go(right) \rightsquigarrow^1 square(right)$$
$$go(left) \rightsquigarrow^1 square(left)$$

as well as more abstract relationships, such as

$$go(x) \rightsquigarrow^1 square(x)$$

The pattern mining algorithm used by ROCCA is detailed in [7]. This is a generic hypergraph pattern miner, not specialized for temporal patterns. In order to mine temporal patterns with it, timestamps are represented as naturals. 0 is presented by Z, 1 by $S(Z)$, 2 by $S(S(Z))$, etc. This provides the needed structure to discover temporal relationships between events. As it currently stands, the pattern miner can efficiently discover mono-action plans. Mining poly-action plans is also possible but has two issues:

1. In the worse case, the computational cost of mining goes up exponentially with the size of the action sequence to mine.
2. The number of observations to accumulate in order to generate cognitive schematics with decent confidences goes up exponentially as well.

The latter is really the most problematic because we cannot buy our way out of it. If each observation takes a certain amount time, determined by the control cycle period in the case of primary observations, then we have to go through them, we cannot speed time up. This is even more true for abstract percepta that can only be observed at periods that are multiples of control cycle periods. Also, in some cases, a particular action sequence may never be observed at all, yet we still would like to have a way to estimate the likelihood of its success. In order to address these limitations and more, we need reasoning.

Temporal Reasoning. ROCCA uses a temporal extension of PLN described in [8] to update existing cognitive schematics obtained by pattern mining, and discover new cognitive schematics by combining existing ones. For instance it can infer poly-action plans by stringing mono-action plans together, as well as generalize or specialize their contexts or goals. Temporal rules integrated into ROCCA include:

1. *Predictive Implication Direct Introduction* to infer the truth value of a predictive implication from direct observations.
2. *Temporal Conditional Conjunction Introduction* to specialize a goal within a plan by considering the conjunction of existing cognitive schematics goals.
3. *Temporal Deduction* to string together small plans to form bigger ones.

The precise semantics of these rules is detailed in [8]. An example of how they are used is presented below.

4 Experiment in a Simple Minecraft Environment

In this experiment, we use Malmo [16] to construct a basic Minecraft world that comprises a house filled with diamonds and a key. The objective of the agent is to retrieve the key, located somewhere in the vicinity of the house, and then unlock the door of the house. Upon unlocking the door, the agent is able to collect a diamond and receive a reward.

The aim of the experiment is to make ROCCA learn from interacting with the Minecraft environment and collect as many diamonds as possible. To make the task easier, the primary actions and perceptions provided by Malmo have been replaced by high level actions such as *go(key)*, *go(house)* and *go(diamonds)*, as well as high level perceptions about the state of the agent such as *outside(house)*, *hold(key)* and the reward for completing a given action, *reward(1)*.

The experiment consists of two iterations of training lasting fifty control cycles each, interleaved by a learning phase of a few hours. During the first iteration, no learning is taking place as the agent has no prior knowledge. The agent randomly explores the environment. Then it enters a learning phase, discovering cognitive schematics via mining and reasoning, subsequently leading the agent to achieve more frequently its goal during the next training phase.

Let us look more closely how ROCCA discovers cognitive schematics. Given the following observations

$$\{\ldots, \textit{Reward(0)@10},\ \textit{outside(house)@10},\ \textit{hold(key)@10},\ \textit{go(house)@10},\ \textit{inside(house)@11},\ \textit{go(diamond)@11},\ \textit{Reward(0)@11},\ \textit{Reward(1)@12}, \ldots\}$$

ROCCA can mine, among many other things, the following cognitive schematic

$$\textit{hold(key)} \wedge \textit{go(house)} \rightsquigarrow^{1} \textit{inside(house)} \stackrel{\text{m}}{=} <0.833, 0.007>$$

Additionally, by applying the temporal conditional conjunction introduction rule on the relevant relationships, such as

$$\textit{outside(house)} \wedge \textit{go(key)} \rightsquigarrow^{1} \textit{outside(house)} \stackrel{\text{m}}{=} <1, 0.007>$$
$$\textit{outside(house)} \wedge \textit{go(key)} \rightsquigarrow^{1} \textit{hold(key)} \stackrel{\text{m}}{=} <1, 0.007>$$

the agent derives

$$\textit{outside(house)} \wedge \textit{go(key)} \rightsquigarrow^{1} \textit{outside(house)} \wedge \textit{hold(key)} \stackrel{\text{m}}{=} <1, 0.007>$$

which, if combined with

$$\textit{outside(house)} \wedge \textit{hold(key)} \wedge \textit{go(house)} \rightsquigarrow^{1} \textit{inside(house)} \stackrel{\text{m}}{=} <0.833, 0.007>$$

can be used by the procedural deduction rule to infer

$$\textit{outside(house)} \wedge \textit{go(key)} \curlywedge^{1} \textit{go(house)} \rightsquigarrow^{2} \textit{inside(house)} \stackrel{\text{m}}{=} <0.833, 0.007>$$

Continuing this reasoning process ultimately results in the discovery of an effective plan that leads to achieving the goal, such as

$$\textit{outside(house)} \wedge \textit{go(key)} \curlywedge^{1} \textit{go(house)} \curlywedge^{1} \textit{go(diamond)} \rightsquigarrow^{3} \textit{reward(1)} \stackrel{\text{m}}{=} <0.833, 0.005>$$

A rigorous evaluation is kept for a future paper, nonetheless our preliminary results indicate that ROCCA successfully learns the necessary cognitive schematics and, as a consequence, accumulates more rewards during the second iteration. In the first iteration the cumulative reward is around 5, then doubles to quadruples during the second iteration, depending on the random seed and other parameters. If ROCCA keeps running after that, the cumulative reward rate keeps going up because the confidences of the cognitive schematics increase, leading to more exploitation and less exploration. One may notice that some plans are not completely reliable, their strengths is below 1. That is because some actions may fail. ROCCA is suited for dealing with uncertainty and has no problem with that. These findings are encouraging but only apply to a very simple environment and may not be indicative of the overall performance of ROCCA. More experiments over more environments are required and will be pursued in future work.

The source code of ROCCA is hosted on Github [2] and a video of this experiment is available on YouTube [1].

5 Conclusion

ROCCA, a system that leverages the OpenCog framework for controlling an agent in uncertain environments has been presented. This agent is in a strong sense fully reasoning-based, from learning to planning. The advantage we believe of such approach, in spite of its current inefficiencies, is to offer greater transparency and foster greater capabilities for meta-learning and self-improvement. As such, we are only at the start of our endeavor. Towards that end, future directions include:

1. Integrate *Economic Attention Networks* [18] for *Attention Allocation*. Record attentional spreading as percepta to learn Hebbian links [18] and improve attention allocation in return.
2. Carry out concept creation and schematization, also called crystallized attention allocation, to speed up repetitive information processing. This done well should also provide a solution to the problem of creating hierarchies of abstract observations and actions.
3. Record more internal processes, not just attentional spreading, as internal percepta to enable deeper forms of introspection.
4. Plan with internal actions, not just external, such as parameter tuning and code rewriting to enable self-improvements.

References

1. ROCCA video demonstration (2022). https://youtu.be/rCvQHrJAD2c
2. ROCCA source code repository (2023). https://github.com/opencog/rocca
3. Bach, J.: MicroPsi 2: the next generation of the MicroPsi framework. In: Bach, J., Goertzel, B., Iklé, M. (eds.) AGI 2012. LNCS (LNAI), vol. 7716, pp. 11–20. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35506-6_2

4. Brockman, G., et al.: OpenAI gym (2016)
5. Cai, Z., et al.: OpenPsi: a novel computational affective model and its application in video games. Eng. Appl. Artif. Intell. **26**, 1–12 (2013). https://doi.org/10.1016/j.engappai.2012.07.013
6. Geisweiller, N.: Partial operator induction with beta distributions. In: Iklé, M., Franz, A., Rzepka, R., Goertzel, B. (eds.) AGI 2018. LNCS (LNAI), vol. 10999, pp. 52–61. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-97676-1_6
7. Geisweiller, N., Goertzel, B.: An inferential approach to mining surprising patterns in hypergraphs. In: Hammer, P., Agrawal, P., Goertzel, B., Iklé, M. (eds.) AGI 2019. LNCS (LNAI), vol. 11654, pp. 59–69. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-27005-6_6
8. Geisweiller, N., Yusuf, H.: Probabilistic logic networks for temporal and procedural reasoning. In: Hammer, P., et al. (eds.) AGI 2023. LNAI, vol. 13921, pp. 85–94. Springer, Cham (2023)
9. Goertzel, B., Ikle, M., Goertzel, I.F., Heljakka, A.: Probabilistic Logic Networks. Springer, New York (2009). https://doi.org/10.1007/978-0-387-76872-4
10. Goertzel, B., et al.: An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in second life. In: Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference, pp. 161–175. IOS Press, NLD (2008)
11. Goertzel, B., Pennachin, C., Geisweiller, N.: Engineering General Intelligence, Part 1: A Path to Advanced AGI via Embodied Learning and Cognitive Synergy. Atlantis Press (2014)
12. Goertzel, B., et al.: Cognitive synergy between procedural and declarative learning in the control of animated and robotic agents using the OpenCogPrime AGI architecture. In: Proceedings of the AAAI Conference on Artificial Intelligence (2011)
13. Hahm, C., Xu, B., Wang, P.: Goal generation and management in NARS. In: Goertzel, B., Iklé, M., Potapov, A. (eds.) AGI 2021. LNCS (LNAI), vol. 13154, pp. 96–105. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-93758-4_11
14. Hammer, P., Lofthouse, T.: 'OpenNARS for applications': architecture and control. In: Goertzel, B., Panov, A.I., Potapov, A., Yampolskiy, R. (eds.) AGI 2020. LNCS (LNAI), vol. 12177, pp. 193–204. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-52152-3_20
15. Hart, D., Goertzel, B.: OpenCog: a software framework for integrative artificial general intelligence. In: Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference, pp. 468–472. IOS Press, NLD (2008)
16. Johnson, M., Hofmann, K., Hutton, T., Bignell, D.: The Malmo platform for artificial intelligence experimentation. In: International Joint Conference on Artificial Intelligence (2016)
17. Leike, J., Lattimore, T., Orseau, L., Hutter, M.: Thompson sampling is asymptotically optimal in general environments. In: Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence. UAI 2016, Arlington, Virginia, USA, pp. 417–426. AUAI Press (2016)
18. Pitt, J., Ikle, M., Sellmann, G., Goertzel, B.: Economic attention networks: associative memory and resource allocation for general intelligence. In: Proceedings of the 2nd Conference on Artificial General Intelligence, pp. 88–93. Atlantis Press, June 2009. https://doi.org/10.2991/agi.2009.19
19. Schmidhuber, J.: Goedel machines: self-referential universal problem solvers making provably optimal self-improvements. ArXiv cs.LO/0309048 (2003)