A General System for Learning and Reasoning in Symbolic Domains

Claes Strannegård[1], Abdul Rahim Nizamani[2], and Ulf Persson[3]

[1] Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg, Sweden and Department of Applied Information Technology, Chalmers University of Technology, Sweden
claes.strannegard@gu.se
[2] Department of Applied Information Technology, University of Gothenburg, Sweden
abdulrahim.nizamani@gu.se
[3] Department of Mathematical Sciences, Chalmers University of Technology, Sweden
ulfp@chalmers.se

Abstract. We present the system O* that operates in arbitrary symbolic domains, including arithmetic, logic, and grammar. O* can start from scratch and learn the general laws of a domain from examples. The main learning mechanism is a formalization of Occam's razor. Learning is facilitated by working within a cognitive model of bounded rationality. Computational complexity is thereby dramatically reduced, while preserving human-level performance. As illustration, we describe the learning process by which O* learns elementary arithmetic. In the beginning, O* knows nothing about the syntax or laws of arithmetic; by the end, it has constructed a theory enabling it to solve previously unseen problems such as "what is $67 * 8$?" and "which number comes next in the sequence $8, 11, 14$?".

Keywords: domain-independent agent, Occam's razor, bounded rationality.

1 Introduction

The goal of artificial general intelligence (AGI) is to build systems with general intelligence at the human level or beyond [1]. Such systems must be able to learn the mechanics of new domains while adapting to arbitrary performance measures in those domains. Developmental (epigenetic) robotics, inspired by developmental psychology, builds agents that develop automatically through interaction with their environment [2]. Research in the field has so far focused on sensorimotor development and social interaction, leaving higher cognitive functions, such as symbolic reasoning, largely unexplored.

A standard textbook in artificial intelligence [3] characterizes an agent as rational if it always acts so that the expected value of its performance measure is maximized. Herbert Simon introduced the notion of *bounded rationality* [4] to designate "rational choice that takes into account the cognitive limitations of the decision maker – limitations of both knowledge and computational capacity".

Inductive program synthesis – in particular, inductive logic programming [5] – is program synthesis from examples of input-output pairs [6]. The analytic approach uses

examples to construct actual programs, while generate-and-test uses examples for testing purposes only. Analytic techniques include anti-unification and recursive relation learning. The computational complexity of the methods used has proven a major obstacle to the use of inductive program synthesis on a larger scale [6].

Cognitive architectures such as Soar [7], ACT-R [8], CHREST [9], and NARS [10] have been used to model various aspects of human cognition. Such architectures commonly use abstract rewrite systems to model computations as rewrite sequences [11]. They often include explicit models of such cognitive resources as working, sensory, declarative, and procedural memory. These cognitive resources are bounded in various ways: e.g., with respect to capacity, duration, and access time [12]. In particular, working memory, which can typically only hold a small number of items (chunks) is a well-known bottleneck for human problem solving [13]. That said, one can easily obtain models of bounded rationality by placing limits on the available cognitive resources.

According to Piaget's developmental theory, children adapt to new information in one of two ways: by assimilation, in which new information fits into existing knowledge structures; and by accommodation, in which new information causes new knowledge structures to form or old structures to be modified [14].

Occam's razor tells us to prefer short and simple explanations, both in science and everyday life. It can be formalized in several ways, some of which (e.g., Kolmogorov and Solomonoff complexity) are not computable, while others (e.g., Levin complexity) are [15]. A computable version of Kolmogorov complexity can be obtained by combining it with traditional complexity classes from complexity theory. Likewise, although the universal AI model AIXI is not computable in its original form, restricted versions have been proposed that are capable of problem solving in e.g. game domains [16].

In this paper, we present a general system, designed to match the learning and reasoning capabilities of unaided humans in symbolic domains. Section 2 presents the system $\mathrm{O}^\star$. Sections 3 and 4 shows how $\mathrm{O}^\star$ can be used for learning and reasoning, respectively. For the sake of concreteness, we consider the special case of elementary arithmetic. Section 5 offers some conclusions.

2 Computational model

In cognitive psychology, computational models should, ideally, perform both (i) at least and (ii) at most at the human level for any given performance measure. For AGI, satisfying (i) is generally sufficient. Indeed, performing *above* the human level is an advantage; not to mention that a unilateral cognitive model satisfying (i) but not (ii) may be easier to construct.

In this paper, we use the following strategy for AGI. Suppose that a human, with bounded cognitive resources, can solve a certain problem. Suppose further that one has a well-designed unilateral cognitive model of this person – one that also has bounded cognitive resources. Then, a solution exists within the model: one that we can find by searching the model. The search strategy can be combined with any heuristic algorithm. By so exploiting the link to human cognition, we hope to mitigate the combinatorial explosion associated with many standard AI algorithms. We now proceed to define our model formally.

Definition 1 (Tag). *A* tag *is a string of ASCII symbols that does not contain any punctuation symbols: in particular, no commas, colons, parentheses, or spaces.*

Tags will be written in monospaced font. For example, `x`, `1` and `Digit` are tags.

Definition 2 (Variable). *A* variable *is a string of the form* `(σ:τ)`*, where* σ *and* τ *are tags.*

For example, `(x:Number)` and `(x:Sent)` are variables.

Definition 3 (Term). *A* term *is a finite tree whose nodes are labeled with tags or variables. Variables may only appear in the leaf nodes.*

Example 1. Here are two examples of terms (about arithmetic and propositional logic):

For purposes of this paper, terms will be presented not as trees but as strings, to make the notation more compact. We use a standard linearization algorithm that inserts parentheses into the strings to reflect the implied tree structure. We use some conventions – e.g. omitting certain parentheses – to make the notation easier to read. The above terms linearize to `(x:Number)*0` and `(x:Sent)||True`, respectively. To go in the reverse direction, we use a standard parsing algorithm. In this way, one can move freely between representations of terms as trees or strings.

Definition 4 (Axiom). *An* axiom *is a triple* (τ, t, t')*, where* τ *is a tag and* t *and* t' *are terms.*

Example 2. Here are some examples of axioms (about arithmetic and propositional logic):

(`Equal`, `1+2`, `3`)

(`Equal`, `(x:Number) * 0`, `0`)

(`Equi`, `(x:Sent) || True`, `True`)

(`Equi`, `(x:Sent) && (y:Sent)`, `(y:Sent) && (x:Sent)`).

Definition 5 (Theory). *A* theory *is a finite set of axioms.*

Definition 6 (Assignment). *An* assignment *is a partial function from variables to terms.*

For instance, $\alpha = \{(\texttt{(x:Number)}, \texttt{1}), (\texttt{(y:Number)}, \texttt{2})\}$ is an assignment. By extension, assignments are defined from terms to terms. Thus, $\alpha(\texttt{(x:Number)+(y:Number)}) = \texttt{1+2}$.

If s is a subtree of t, we write $t(s)$. Moreover, if s' is an arbitrary tree, we write $t(s'/s)$ to denote the result of replacing all occurrences of s in t by s'.

Definition 7 (Rewrite). *Suppose* (τ, t_1, t_2) *is an axiom. Then, we write*

$$\frac{t(t')}{t(t''/t')}\ (\tau, t_1, t_2)$$

if there is an assignment α *such that* $\alpha(t_1) = t'$ *and* $\alpha(t_2) = t''$*. The conclusion* $t(t'')$ *follows from the premise* $t(t')$ *by* Rewrite*.*

Example 3. Here is an example application of Rewrite (with α as above):

$$\frac{\texttt{1+2}}{\texttt{2+1}}\ (\texttt{Equal}, \texttt{(x:Number)+(y:Number)}, \texttt{(y:Number)+(x:Number)})$$

Definition 8 (Computation). *A* computation *is a sequence of terms* $(t_0, \ldots, t_n)$ *such that for all* $i < n$*,* t_{i+1} *follows from* t_i *by application of Rewrite.*

We write computations vertically, from top to bottom, with the relevant axioms shown at the transitions.

Example 4. Here is an example of a computation in the domain of arithmetic:

$$\begin{array}{c} \texttt{(1+2)*3} \\ \hline \texttt{3*3} \\ \hline \texttt{9} \end{array}\ \begin{array}{l} (\texttt{Equal}, \texttt{1+2}, \texttt{3}) \\ (\texttt{Equal}, \texttt{3*3}, \texttt{9}) \end{array}$$

Example 5. Below is an example of a computation in the domain of propositional logic. Here `x`, `y`, and `z`, are abbreviations of `(x:Sent)`, `(y:Sent)`, and `(z:Sent)`, respectively. Intuitively, the computation is a proof of the tautology $(p \to q) \vee p$. This is because all of the axioms used in the computation preserve logical equivalence.

$$\begin{array}{c} \texttt{(p => q) || p} \\ \hline \texttt{((not p) || q) || p} \\ \hline \texttt{(q || (not p)) || p} \\ \hline \texttt{q || ((not p) || p)} \\ \hline \texttt{q || True} \\ \hline \texttt{True} \end{array}\ \begin{array}{l} (\texttt{Equi}, \texttt{x => y}, \texttt{(not x) || y}) \\ (\texttt{Equi}, \texttt{x || y}, \texttt{y || x}) \\ (\texttt{Equi}, \texttt{(x || y) || z}, \texttt{x || (y || z)}) \\ (\texttt{Equi}, \texttt{(not x) || x}, \texttt{True}) \\ (\texttt{Equi}, \texttt{x || True}, \texttt{True}) \end{array}$$

Definition 9 (Agent). *An* agent *is a tuple* (T, C, W, L, D)*, where*

- *T is a theory (representing beliefs in declarative memory);*
- *C is a set of terms (representing concepts in declarative memory);*
- *W is a natural number (representing working memory capacity);*
- *L is a natural number (representing assimilation capacity); and*
- *D is a natural number (representing accommodation capacity).*

Definition 10 (Term size). *Given a term* t*,* $s(t)$ *is the number of nodes of* t*.*

The measure $s(t)$ is a simple model of the load of t on working memory. We also define $s(ax) = s(t) + s(t')$ for axioms $ax = (\tau, t, t')$ and $s(T) = \Sigma\{s(ax) : ax \in T\}$ for theories T.

Definition 11 (Agent computation). *Let* $A = (T, C, W, L, D)$ *be an agent. An* A-computation *is a sequence of terms* $(t_0, \ldots, t_n)$ *with restrictions on*

- *terms:* $t_i \in C$*, for all* $i < n$*;*
- *transitions:* t_{i+1} *is obtained from* t_i *using Rewrite and an axiom from* T*, for all* $i < n$*;*
- *width:* $s(t_i) \leq W$*, for all* $i \leq n$*; and*
- *length:* $n \leq L$*.*

The definition aims to capture those computations within reach of an unaided human with belief set T, concept set C, working memory capacity W, assimilation capacity L, and accommodation capacity D (*cf.* Definition 19).

Observation 1 *For any agent A, the set of A-computations is finite.*

This holds for combinatorial reasons, since all resources of A are bounded. Any A-computation can be obtained by inserting terms belonging to a finite set C into a frame of finite length L and width W.

Definition 12 (Induction problem). *An* induction problem *(IP) is a finite set of items, where an* item *is a tuple* (τ, t, t', u) *such that* τ *is a tag,* t *and* t' *are variable-free terms, and* u *is an integer (utility).*

Definition 13 (Item computation). *The agent A* computes *the item* (τ, t, t', u) *if there is an A-computation c from* t *to* t' *that uses only axioms with the tag* τ *and has the property that, for every assignment* $\alpha((\sigma : \tau'')) = t''$ *occurring in c, there is a (type-checking) A-computation from* t'' *to* τ'' *that uses only axioms with the tag* `Type`*.*

Example 6. Suppose that the following are A-computations –the right one being a type-checking computation. Then A computes the item $(\texttt{Equal}, \texttt{2+0}, 2, 1)$.

$$\frac{\texttt{2+0}}{\texttt{2}}\ (\texttt{Equal}, \texttt{(x:Digit)+0}, \texttt{(x:Digit)}) \qquad\qquad \frac{\texttt{2}}{\texttt{Digit}}\ (\texttt{Type}, \texttt{2}, \texttt{Digit})$$

Definition 14 (Performance). *The* performance *of agent A on induction problem I is the number*

$$\Sigma\{u : (\tau, t, t', u) \in I \text{ and } A \text{ computes } (\tau, t, t', u)\}.$$

The convention that $\Sigma\emptyset = 0$ *ensures that the performance is always defined.*

Observation 2 *The performance measure is computable.*

Next, let us introduce the four operators to be used for constructing theories.

Definition 15 (Crossover). *The term* t'' *is obtained from the terms* t *and* t' *by* crossover *if* t'' *can be obtained from* t *by replacing a subtree of* t *by a subtree of* t'*.*

For example, $1 + (3 * 4)$ is a crossover of $1 + 2$ and $3 * 4$.

Definition 16 (Abstraction). *The axiom* (τ, t, w) *is obtained from the item* (τ, t', w', u) *by* abstraction *if* $\alpha(t) = t'$ *and* $\alpha(w) = w'$*, for some assignment* α*.*

Example 7. $(\texttt{Equal}, \texttt{(x:Aterm)+(y:Aterm)}, \texttt{(y:Aterm)+(x:Aterm)})$ is obtained from the item $(\texttt{Equal}, \texttt{1+2}, \texttt{2+1}, 1)$ by abstraction.

Definition 17 (Recursion). *The axiom ax is obtained from the item* $(\tau, f(t), t', u)$ *by* recursion *if* $ax = (\tau, f(w), w')$*, where* w' *contains* $f(w'')$*, and* w *and* w'' *contain the same variable.* w *and* w' *are formed by* crossover.

Example 8. The axiom $(\texttt{Equal}, \texttt{f((x:Aterm)+1)}, \texttt{f((x:Aterm))*2})$ is obtained from the item $(\texttt{Equal}, \texttt{f(0)}, 1, 1)$ by recursion.

Definition 18 (Memorization). *The axiom ax is obtained by* memorization *from the item* (τ, t, t', u)*, where* $u > 0$*, if* $ax = (\tau, t, t')$.

Example 9. The axiom $(\texttt{Equal}, \texttt{1+2}, 3)$ is obtained from the item $(\texttt{Equal}, \texttt{1+2}, 3, 1)$ by memorization.

Definition 19 (Occam function). *The* Occam function *takes agent* $A_n = (T_n, C_n, W, L, D)$ *and IP* I_n *as input and outputs agent* $A_{n+1} = (T_{n+1}, C_{n+1}, W, L, D)$*, as specified below.*

Let $C_{n+1} = C_n \cup \Gamma$*, where* Γ *is obtained from* I_n *by taking subtrees of terms that appear in items of* I_n*, and replacing one or more leaf nodes of those subtrees by variables taken from a set generated from* C_n.

To define T_{n+1}*, first form the set* Δ *consisting of:*

- *All axioms ax such that* $s(ax) \leq D$*, whose terms are obtained from* C_{n+1} *by crossover;*
- *All axioms obtained from items of* I_n *by abstraction;*
- *All axioms ax such that* $s(ax) \leq D$ *that are obtained from items of* I_n *by recursion, using terms of* C_{n+1}*; and*
- *All axioms obtained from items of* I_n *by memorization.*

Then, form the set $\Delta' \subseteq \Delta$*, whose axioms satisfy a few additional conditions: e.g., all variables must appear in both terms of the axiom, or not at all.*

Next, form the set $\Delta'' \subseteq \Delta'$ *containing at most 3 axioms, using the preference order:*

1. *The performance of* $(T_n \cup \Delta'', C_{n+1}, W, L, D)$ *on* I_n *is as large as possible;*
2. $s(\Delta'')$ *is as small as possible;*
3. Δ'' *has a maximum number of variable tokens;*
4. Δ'' *has a minimum number of variable types;*
5. Δ'' *is as lexicographically small as possible.*

Finally, let $T_{n+1} = T_n \cup \Delta''$.

To illustrate the difference between type and token in this case, we can consider the term `(x:Number)*(x:Number) + 2`, which contains one variable type and two variable tokens. Condition 1 aims to maximize utility. Condition 2 is a formalization of Occam's razor. Condition 3 ensures that variables are preferred over constants and Condition 4 that variables are reused whenever possible. Condition 5, finally, guarantees that the output is always uniquely defined.

Observation 3 *The Occam function is computable.*

This follows since there are only finitely many agents and associated computations to check. We implemented the system O* comprising approximately 2,500 lines of code in the functional programming language Haskell. O* can be initialized with any agent A_0. At stage n, O* takes IP I_n as input and updates A_n to A_{n+1}. O*'s update function is a version of the Occam function with the following additions.

To reduce search complexity, O* applies the Occam function in multiple steps. First it forms the set Δ by *crossover* and iterates over lengths of candidates (1 to D). If *crossover* fails to find an optimal candidate for Δ'', O* uses *abstraction* and then *recursion*, and finally *memorization* if all else fails. The search for Δ'' from Δ' is exhaustive, while the search for A-computations uses the A^* algorithm with the goal of finding the shortest computations.

Crossover produces a large number of axioms. Therefore, a small set of filters is applied for reducing the complexity while still maintaining the generality of the resulting set Δ. These filters include the following: an axiom should be new (not already in the theory); it should not have a variable in the right term that is absent from the left term (cf. pure functions); it should contain at most two wildcards (variables appearing in the left term but not in the right term), e.g., $(\texttt{Equal}, \texttt{(x:Aterm) * 0}, \texttt{0})$. Moreover, variable assignments, e.g., $(\texttt{Equal}, \texttt{(x:Aterm)}, \texttt{1})$, are not allowed.

3 Learning

In this section we illustrate how O* can learn in the special case of elementary arithmetic. All problems considered below were solved by O* running on a standard computer.

Example 10. Define A_0 by letting $T_0 = \emptyset$, $C_0 = \emptyset$, $W = 8$, $L = 10$, and $D = 6$. Suppose I_0 consists of the items

$$(\texttt{Type}, \texttt{0}, \texttt{Digit}, 1) \tag{1}$$
$$(\texttt{Type}, \texttt{1}, \texttt{Digit}, 1) \tag{2}$$
$$(\texttt{Type}, \texttt{2}, \texttt{Digit}, 1). \tag{3}$$

Then, C_1=$\{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{Digit}\}$ and T_1 consists of the axioms

$$(\texttt{Type}, \texttt{0}, \texttt{Digit}) \tag{4}$$
$$(\texttt{Type}, \texttt{1}, \texttt{Digit}) \tag{5}$$
$$(\texttt{Type}, \texttt{2}, \texttt{Digit}). \tag{6}$$

Item (1) is A_1-computable as follows:

$$\frac{\texttt{0}}{\texttt{Digit}} \ (4)$$

The other two items can be computed similarly. Intuitively, A_1 has memorized that 0, 1, and 2 are digits.

Example 11. Let A_1 be as above. Suppose I_1 consists of the items

$$(\texttt{Type}, \texttt{1}, \texttt{Number}, 1) \tag{7}$$
$$(\texttt{Type}, \texttt{1\#2}, \texttt{Number}, 1) \tag{8}$$
$$(\texttt{Type}, \texttt{1\#(2\#1)}, \texttt{Number}, -1). \tag{9}$$

The symbol # can be interpreted as a juxtaposition operator: e.g., 1#2 is interpreted as 12. Now $C_2 - C_1 = \{\texttt{1\#2}, \texttt{2\#1}, \texttt{1\#(2\#1)}, \texttt{Number}\}$ and $T_2 - T_1$ consists of the axioms

$$(\texttt{Type}, \texttt{Digit}, \texttt{Number}) \tag{10}$$
$$(\texttt{Type}, \texttt{Number\#Digit}, \texttt{Number}). \tag{11}$$

Item (7) is A_2-computable using the axioms (5) and (10). Item (8) is A_2-computable as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\texttt{1\#2}}{\texttt{Digit\#2}}\,(5)}{\texttt{Number\#2}}\,(10)}{\texttt{Number\#Digit}}\,(6)}{\texttt{Number}}\,(11)$$

Item (9) is not A_2-computable. It is not hard to see that T_2 is too weak to compute this item. Therefore, $\mathrm{O}^\star$ terminates the search and concludes that the item is not computable. The performance of A_2 on I_1 is 2, which is optimal. If item (9) were not included in I_1, then $T_2 - T_1$ would include the axiom:

$$(\texttt{Type}, \texttt{Number\#Number}, \texttt{Number})$$

instead of the axiom (11). Intuitively, A_2 knows that numbers are sequences of digits.

Example 12. Let A_2 be as above. Suppose I_2 consists of the items

$$(\texttt{Type}, \texttt{1}, \texttt{Aterm}, 1) \tag{12}$$
$$(\texttt{Type}, \texttt{1+2}, \texttt{Aterm}, 1). \tag{13}$$

Then, $T_3 - T_2$ consists of the axioms

$$(\texttt{Type}, \texttt{Number}, \texttt{Aterm}) \tag{14}$$
$$(\texttt{Type}, \texttt{Aterm+Aterm}, \texttt{Aterm}). \tag{15}$$

Item (13) is readily A_3-computable using the axioms (10), (14), and (15).

Example 13. Let A_3 be as above. Suppose I_3 consists of the item

$$(\texttt{Type}, \texttt{1*2}, \texttt{Aterm}, 1). \tag{16}$$

Then, $T_4 - T_3$ consists of the axiom

$$(\texttt{Type}, \texttt{Aterm*Aterm}, \texttt{Aterm}). \tag{17}$$

Now the system has learned the syntactic notions of digit, number, and arithmetical term. Next it will learn some algebraic laws.

Example 14. Let A_4 be as above. Suppose I_4 consists of the items

$$(\texttt{Equal}, \texttt{1+0}, 1, 1) \tag{18}$$

$$(\texttt{Equal}, \texttt{1+1}, 1, -1). \tag{19}$$

Then, $T_5 - T_4$ consists of the axiom:

$$(\texttt{Equal}, \texttt{(x:Aterm)+0}, \texttt{(x:Aterm)}). \tag{20}$$

Item (18) is A_5-computable as follows:

$$\frac{\texttt{1+0}}{1} \ (20)$$

The straightforward type-checking computation uses the axioms (2), (10), and (14). Item (19) is not A_5-computable, since no axiom of T_5 matches `1+1`. If item (19) were not included in I_4, then $T_5 - T_4$ would consist of

$$(\texttt{Equal}, \texttt{(x:Aterm)+(y:Aterm)}, \texttt{(x:Aterm)}).$$

Example 15. Let A_5 be as above. Suppose I_5 consists of the item

$$(\texttt{Equal}, \texttt{0+1}, \texttt{1+0}, 1). \tag{21}$$

Then, $T_6 - T_5$ consists of the axiom

$$(\texttt{Equal}, \texttt{(x:Aterm)+(y:Aterm)}, \texttt{(y:Aterm)+(x:Aterm)}). \tag{22}$$

Item (21) is A_6-computable as follows:

$$\frac{\texttt{0+1}}{\texttt{1+0}} \ (22)$$

The two type-checking computations are, once again, straightforward.

Example 16. Starting from A_6, we may continue in the same way and eventually arrive at an agent, A_n, which contains (a subset of) the theory `BASE`, defined in Appendix A. In particular T_n contains the axioms

$$(\texttt{Equal}, \texttt{8+3}, \texttt{1\#1}) \tag{23}$$

$$(\texttt{Equal}, \texttt{f(1)}, \texttt{f(0+1)}). \tag{24}$$

4 Reasoning

In this section, we illustrate how O* is able to reason about the domains it has learned about and solve previously unseen problems of deduction and induction. For simplicity, we continue to work in the arithmetic domain. First, we will consider the deduction problem of computing $67 * 8$.

Example 17. Let A_n be as above. Let I_n be given by

$$(\texttt{Equal}, \texttt{(6\#7)*8}, \texttt{(5\#3)\#6}, 1). \tag{25}$$

Then, $T_{n+1} = T_n$ and the item is computable by A_{n+1} as follows:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\texttt{(6\#7)*8}}{\texttt{(6*8)\#(7*8)}}}{\texttt{(6*8)\#(5\#6)}}}{\texttt{((6*8)+5)\#6}}}{\texttt{((4\#8)+5)\#6}}}{\texttt{(4\#(8+5))\#6}}}{\texttt{(4\#(1\#3))\#6}}}{\texttt{((4+1)\#3)\#6}}}{\texttt{(5\#3)\#6}}$$

Next, we will consider the induction problem of finding the next number in the sequence $8, 11, 14$.

Example 18. Let A_{n+1} be as above and let I_{n+1} be given by

$$(\texttt{Equal}, \texttt{f(0)}, 8, 1) \tag{26}$$
$$(\texttt{Equal}, \texttt{f(1)}, \texttt{1\#1}, 1) \tag{27}$$
$$(\texttt{Equal}, \texttt{f(2)}, \texttt{1\#4}, 1). \tag{28}$$

Then, $T_{n+2} - T_{n+1}$ consists of the axioms

$$(\texttt{Equal}, \texttt{f(0)}, 8) \tag{29}$$
$$(\texttt{Equal}, \texttt{f((x:Aterm)+1)}, \texttt{f((x:Aterm))+3}). \tag{30}$$

For instance, item (27) is computable by A_{n+2} as follows:

$$\cfrac{\cfrac{\cfrac{\texttt{f(1)}}{\texttt{f(0+1)}}\,(24)}{\texttt{f(0)+3}}\,(30)}{\cfrac{\texttt{8+3}}{\texttt{1\#1}}\,(29)}\,(23)$$

Now, A_{n+2} can compute `f(3)` to obtain `1#7` and thus solve the number-sequence problem "correctly".

5 Conclusions

We have described the general system O* for learning and reasoning in symbolic domains. O* differs from standard AI models by being domain independent and by containing a unilateral cognitive model whose purpose is to reduce computational complexity, while keeping performance at the human level or above. In this way, the combinatorial-explosion problem, arising in e.g. inductive logic programming, automatic theorem

proving, and grammar induction, is mitigated. O* is able to learn the mechanics of new symbolic domains from scratch. It is general purpose: it has nothing built in that is specific to arithmetic or any other particular domain.

This paper showed that O* can learn elementary arithmetic from scratch. After learning, it could solve both deductive problems like computing $67 * 8$ and inductive problems like computing the next number in the sequence $8, 11, 14$. In [17], it was shown that an agent similar to A_n is able to perform above the average human level on number sequence problems appearing in IQ tests. The paper [18] showed that O* can also learn propositional logic from scratch. After learning, it was able to perform above the average human level on propositional logic problems where the task is to distinguish tautologies from non-tautologies.

At present, the main heuristic of O* is to confine the search for solutions to the cognitive model. This heuristic is unsophisticated; nevertheless, it is sufficient for reaching human-level performance in certain domains. It can potentially be combined with more traditional heuristics to improve performance further.

We find the approach to AGI as proposed in this paper promising. More research is needed, however, to determine its applicability, scalability, and robustness.

Acknowledgement

This research was supported by The Swedish Research Council, Grant 421-2012-1000.

A Arithmetic theory

The following are the axioms of the arithmetic theory `BASE`. Here `x`, `y`, and `z` are abbreviations of `(x:Aterm)`, `(y:Aterm)`, and `(z:Aterm)`, respectively.

- `(Type, 0, Digit)`
- `(Type, 1, Digit)`
- ...
- `(Type, 9, Digit)`
- `(Type, Digit, Number)`
- `(Type, Number#Digit, Number)`
- `(Type, Number, Aterm)`
- `(Type, Aterm+Aterm, Aterm)`
- `(Type, Aterm*Aterm, Aterm)`
- `(Equal, x+0, x)`
- `(Equal, 0+x, x)`
- `(Equal, 1+1, 2)`
- `(Equal, 1+2, 3)`
- ...
- `(Equal, 9+9, 1#8)`
- `(Equal, x*0, 0)`
- `(Equal, 0*x, 0)`
- `(Equal, x*1, x)`
- `(Equal, 1*x, x)`
- `(Equal, 2*2, 4)`
- `(Equal, 2*3, 6)`
- ...
- `(Equal, 9*9, 8#1)`
- `(Equal, x+y, y+x)`
- `(Equal, x+(y+z), (x+y)+z)`
- `(Equal, (x#0)+y, x#y)`
- `(Equal, (x#y)+z, x#(y+z))`
- `(Equal, x#(y#z), (x+y)#z)`
- `(Equal, x*(y+z), (x*y)+(x*z))`
- `(Equal, (x#y)*z, (x*z)#(y*z))`
- `(Equal, f(1), f(0+1))`
- `(Equal, f(2), f(1+1))`
- `(Equal, f(3), f(2+1))`

References

1. Wang, P., Goertzel, B.: Theoretical Foundations of Artificial General Intelligence. Atlantis Press (2012)
2. Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., Thelen, E.: Autonomous Mental Development by Robots and Animals. Science **291**(5504) (2001) 599–600
3. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall series in artificial intelligence. Prentice Hall (2010)
4. Simon, H.A.: Models of Bounded Rationality: Empirically Grounded Economic Reason. Volume 3. MIT press (1982)
5. Muggleton, S., Chen, J.: Guest editorial: special issue on Inductive Logic Programming (ILP 2011). Machine Learning (2012) 1–2
6. Kitzelmann, E.: Inductive Programming: A Survey of Program Synthesis Techniques. In: Approaches and Applications of Inductive Programming. Springer (2010)
7. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An Architecture for General Intelligence. Artificial Intelligence **33**(3) (1987) 1–64
8. Anderson, J.R., Lebiere, C.: The atomic components of thought. Lawrence Erlbaum, Mahwah, N.J. (1998)
9. Gobet, F., Lane, P.: The CHREST Architecture of Cognition: The Role of Perception in General Intelligence. In: Artificial General Intelligence 2010, Lugano, Switzerland, Atlantis Press (2010)
10. Wang, P.: From NARS to a Thinking Machine. In: Proceedings of the 2007 Conference on Artificial General Intelligence, Amsterdam, IOS Press (2007) 75–93
11. Bezem, M., Klop, J.W., de Vrijer, R.: Term Rewriting Systems. Cambridge University Press (2003)
12. Smith, E.E., Kosslyn, S.M.: Cognitive Psychology: Mind and Brain. Upper Saddle River, NJ: Prentice-Hall (2006)
13. Toms, M., Morris, N., Ward, D.: Working Memory and Conditional Reasoning. The Quarterly Journal of Experimental Psychology **46**(4) (1993) 679–699
14. Piaget, J.: La construction du réel chez l'enfant. Delachaux & Niestlé (1937)
15. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and Its Applications. Texts in computer science. Springer (2009)
16. Veness, J., Ng, K.S., Hutter, M., Uther, W., Silver, D.: A Monte-Carlo AIXI approximation. Journal of Artificial Intelligence Research **40**(1) (2011) 95–142
17. Strannegård, C., Nizamani, A.R., Sjöberg, A., Engström, F.: Bounded Kolmogorov complexity based on cognitive models. In Kühnberger, K.U., Rudolph, S., Wang, P., eds.: Proceedings of AGI 2013, Beijing, China, Springer (2013)
18. Nizamani, A.R., Strannegård, C.: Learning Propositional Logic From Scratch. In: The 28th annual workshop of the Swedish Artificial Intelligence Society (SAIS), 2014. (in press)