Symbolic Reasoning with Bounded Cognitive Resources

Claes Strannegård (claes.strannegard@gu.se)
Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg
Department of Applied Information Technology, Chalmers University of Technology

Abdul Rahim Nizamani (abdulrahim.nizamani@gu.se)
Department of Applied Information Technology, University of Gothenburg

Fredrik Engström (fredrik.engstrom@gu.se)
Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg

Olle Häggström (olleh@chalmers.se)
Department of Mathematical Sciences, Chalmers University of Technology

Abstract

We present a multi-domain computational model for symbolic reasoning that was designed with the aim of matching human performance. The computational model is able to reason by deduction, induction, and abduction. It begins with an arbitrary theory in a given domain and gradually extends this theory as new regularities are learned from positive and negative examples. At the core of the computational model is a cognitive model with bounded cognitive resources. The combinatorial explosion problem, which frequently arises in inductive learning, is tackled by searching for solutions inside this cognitive model only. By way of example, we show that the computational model can learn elements of two different domains, namely arithmetic and English grammar.

Keywords: symbolic reasoning; bounded resources

Introduction

Artificial intelligence (AI) concerns computer models with intelligence that ideally matches or exceeds that of humans. Strong AI, a.k.a. artificial general intelligence, targets general intelligence, whereas weak AI targets domain-specific intelligence. Weak AI has attained enormous success over the last decades, whereas strong AI has yet to be achieved (Wang & Goertzel, 2012; Kühnberger et al., 2013). The only known cognitive system with general intelligence is the human brain; thus, researchers in strong AI are seeking inspiration from neuroscience and cognitive psychology.

In this paper, we propose a strong AI model for the case of symbolic reasoning. The proposed model is able to reason by induction, deduction, and abduction. The model includes a simplified cognitive model with explicit representations of several cognitive resources, including declarative and working memory. Our rationale for including a cognitive model in our model for strong AI is to exploit the limitations of human cognitive resources to decrease the computational complexity. This paper constitutes a continuation and generalization of our earlier work on models of deduction and induction with bounded resources (Strannegård, Engström, et al., 2013; Strannegård, Nizamani, et al., 2013).

In this paper, we consider symbolic reasoning, which involves discrete signals, in contrast to sub-symbolic reasoning, which involves analogue signals. Peirce (1958) distinguished between three types of symbolic reasoning: deduction, induction, and abduction. These processes have been described as methods to draw rational conclusions, generalize from experience, and make plausible assumptions, respectively (Garcez & Lamb, 2011).

There is an extensive body of literature on symbolic reasoning in logic, psychology, computer science, and linguistics. In logic, reasoning is studied in the form of formal proofs in proof theory (Troelstra & Schwichtenberg, 2000) and in psychology, reasoning is studied, e.g. in the mental logic and mental model traditions (Adler & Rips, 2008).

One research field in computer science that studies reasoning is automatic theorem proving (Robinson & Voronkov, 2001). Another field is machine learning, including Bayesian inference, which considers inferences in a setting where probabilistic data are available (Tenenbaum et al., 2011). A third research field involving reasoning is inductive logic programming (Muggleton & Chen, 2012) and, more generally, inductive program synthesis, where the goal is to develop programs from examples consisting of input-output pairs (Kitzelmann, 2010). The two main approaches of inductive program synthesis are the analytical approach, which uses examples as a basis for constructing programs, and the generate-and-test approach, which uses examples for testing purposes only. Analytical techniques include anti-unification and recursive relation learning. A representative system of the generate-and-test approach, MAGICHASKELLER searches among Haskell programs that may include higher-order functions (Katayama, 2005). The systems FLIP (Ferri-Ramírez et al., 2001) and ADATE (Olsson, 1998) blend the two approaches in an inductive logic programming setting. The system ADATE also uses evolutionary methods to generate programs. A major obstacle to program synthesis on a larger scale is the computational complexity of the state-of-the-art methods (Kitzelmann, 2010).

Reasoning is also studied in computational linguistics, particularly in grammar induction (Clark & Lappin, 2010), where classes of languages are learned from positive and negative examples. Computational complexity is a major issue in this field as well.

A particular line of research on reasoning can be traced back to *Occam's razor*, the principle of preferring simple and short explanations in science and everyday life. There are several complexity measures that can be regarded as formal versions of Occam's razor. Some of these, e.g., Kolmogorov complexity and Solomonoff complexity, are not computable, whereas others, e.g., Levin complexity, are computable (Li & Vitányi, 2009). Some of the computable versions are obtained by combining Kolmogorov complexity with traditional complexity classes pertaining to the time and space used by Turing machines.

The universal AI model AIXI is based on Solomonoff complexity and is therefore not computable in its original form. However, this model also exists in restricted versions that are computable and capable of practical problem solving, e.g., in game domains (Veness et al., 2011).

Different aspects of reasoning have been modeled in cognitive architectures, such as Soar (Laird et al., 1987), ACT-R (Anderson & Lebiere, 1998), CHREST (Gobet & Lane, 2010), and NARS (Wang, 2007). Cognitive architectures commonly model computations as rewrite sequences in abstract rewrite systems (Bezem et al., 2003). Moreover, they often include explicit models of cognitive resources, including working, sensory, declarative, and procedural memory. These cognitive resources are bounded in various ways, e.g., with respect to capacity, duration, and access time (Kosslyn & Smith, 2006). In particular, working memory can typically only hold a small number of items, or chunks, and is a well-known bottleneck of human problem solving (Toms et al., 1993).

Cognitive resources are required for computing and learning. According to Piaget, there are two ways of adapting to new information: *assimilation*, in which new pieces of information are fitted into existing knowledge structures, and *accommodation*, in which the new information pieces cause new knowledge structures to be formed or old structures to be modified (Piaget, 1937).

Computational model

Computational models that are used in cognitive psychology should ideally perform on the human level with respect to any performance measure. Thus, the models should perform (i) at least on the human level and (ii) at most on the human level. For AI, however, satisfying (i) is generally sufficient. In fact, performing above the human level is only an advantage. Thus, constructing a *unilateral* cognitive model that satisfies (i) but not necessarily (ii) may be sufficient for the purpose of AI and also considerably easier than constructing a psychologically plausible model that satisfies both (i) and (ii).

Our computational model includes such a unilateral cognitive model, and our rationale is as follows. Suppose a human (with bounded cognitive resources) can find a solution to a certain problem. A solution to the problem should then exist inside a suitable cognitive model of this human (also with bounded resources). Thus, we can search for solutions exclusively among those solution candidates that use no more cognitive resources than are available. This search strategy can be combined with any heuristic search algorithm. In this manner, we may accelerate the search considerably, thus avoiding the combinatorial explosion that is associated with many standard AI algorithms.

Our computational model also includes a performance measure for agents that was inspired by the notion of biological fitness (the ability of an organism to survive and reproduce). More precisely, we assume that a notion of reward and punishment exists in the background, similar to the scheme in reinforcement learning models. Thus, each example will have an associated utility, which may be positive (reward) or negative (punishment). We now proceed to define our computational model formally.

Definition 1 A *language* is a set of strings that is generated by some finite context-free grammar.

The elements of a language L are called *L-terms*. Two examples of languages, whose grammars are expressed in the Backus-Naur form, are given below.

Example 1 Below is a definition of the language `Stream`, consisting of arbitrary streams of words over a five-word vocabulary. We use `S` as the start symbol of grammars.

```
Word = Alice | Bob | plays | crawls | OK
S = Word | Word S
```

For example, `Alice crawls` and `plays plays` are `Stream`-terms.

Example 2 A definition of the language `Arith`, consisting of simple arithmetical expressions, is shown below.

```
Digit = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Num = Digit | Num Digit
Var = x | y | z
Unary = f
Binary = + | g
Chunk = f(0) | f(x) | f(x+1) | g(x,0) |
     g(x,y) | g(x,y+1)
S = Num | Var | Unary S | Binary S S | Chunk
```

For example, `2+4` and `f(x+1)` are (pretty-printed) `Arith`-terms. Strictly speaking, the syntactic category `Chunk` is redundant here. Our reason for including it nevertheless is to model memory chunks.

Definition 2 An *L-axiom* is an expression of the form $t \twoheadrightarrow t'$, where t and t' are L-terms.

For example, `x+y` $\twoheadrightarrow$ `y+x` is an `Arith`-axiom and `Alice crawls` $\twoheadrightarrow$ `OK` is a `Stream`-axiom.

Definition 3 An *L-theory* is a finite set of L-axioms.

The purpose of theories is to serve as building blocks of computations, as explained below.

Definition 4 Suppose that t and t' are L-terms such that t' is a substring of t. Then, we say that t' is a *subterm* of t and write $t(t')$.

Languages may include the syntactic category `Var`, which specifies the *variables* of L. For instance, `Arith` contains the variables `x`, `y`, and `z`.

Definition 5 An *L-substitution* is a partial function σ that maps variables of L to terms of L.

For instance, $\sigma = \{(\texttt{x}, 2), (\texttt{y}, 3)\}$ is an `Arith`-substitution. By extension, we will discuss substitutions that map terms to terms. In the same example, we obtain $\sigma(\texttt{x+y}) = \texttt{2+3}$. We are now ready to define our only computation rule.

Definition 6 Let L be a language, and let T be an L-theory containing the axiom $t' \twoheadrightarrow t''$. Additionally, suppose that $t(t_1)$ and $t(t_2)$ are L-terms such that $\sigma(t') = t_1$ and $\sigma(t'') = t_2$ for some L-substitution σ. Then, we write

$$\frac{t(t_1)}{t(t_2)}\ t' \twoheadrightarrow t''$$

and state that the conclusion (the term below the line) follows from the premise (the term above the line) by *T-rewriting*.

Example 3 Suppose that T is a theory in the language `Arith`, defined in Example 2, and $\texttt{x+y} \twoheadrightarrow \texttt{y+x} \in T$. Then, the following is an application of T-rewriting (with σ as above):

$$\frac{\texttt{2+3}}{\texttt{3+2}}\ \texttt{x+y} \twoheadrightarrow \texttt{y+x}$$

Definition 7 Let L be a language, and let T be an L-theory. A *T-computation* is a sequence of terms $(t_0, \ldots, t_n)$ such that for all $i \leq n$, t_{i+1} follows from t_i by an application of T-rewriting.

We will write computations in a vertical style, with the first term of the computation on top and the last term at the bottom.

Example 4 Below is a computation in a simple theory of English grammar. Intuitively, the computation demonstrates that the word stream `Bob plays` is grammatically correct.

$$\cfrac{\cfrac{\cfrac{\texttt{Bob plays}}{\texttt{Bob crawls}}\ \texttt{plays} \twoheadrightarrow \texttt{crawls}}{\texttt{Alice crawls}}\ \texttt{Bob} \twoheadrightarrow \texttt{Alice}}{\texttt{OK}}\ \texttt{Alice crawls} \twoheadrightarrow \texttt{OK}$$

Example 5 Below is a computation in a simple theory of arithmetic. Intuitively, the computation is a calculation of the term `(2+4)*(6+1)`. In fact, all of the axioms used here preserve equality.

$$\cfrac{\cfrac{\cfrac{\texttt{(2+4)*(6+1)}}{\texttt{6*(6+1)}}\ \texttt{2+4} \twoheadrightarrow \texttt{6}}{\texttt{6*7}}\ \texttt{6+1} \twoheadrightarrow \texttt{7}}{\texttt{42}}\ \texttt{6*7} \twoheadrightarrow \texttt{42}$$

Example 6 Below is a computation in a simple theory of propositional logic. Intuitively, the computation is a proof of the tautology $(x \Rightarrow y) \vee x$. In fact, all of the axioms used here preserve logical equivalence.

$$\cfrac{\cfrac{\cfrac{\cfrac{\texttt{(x => y) || x}}{\texttt{(not x || y) || x}}\ \texttt{(x => y)} \twoheadrightarrow \texttt{(not x || y)}}{\texttt{(y || not x) || x}}\ \texttt{x || y} \twoheadrightarrow \texttt{y || x}}{\texttt{y || (not x || x)}}\ \texttt{(x || y) || z} \twoheadrightarrow \texttt{x || (y || z)}}{\texttt{y || True}}\ \texttt{x || not x} \twoheadrightarrow \texttt{True}}{\texttt{True}}\ \texttt{x || True} \twoheadrightarrow \texttt{True}$$

Example 7 Below is a computation in a simple theory of lists. This computation is expressed in a language similar to the Haskell programming language. Intuitively, the computation demonstrates that the result of applying the reversing function `rev` to the list `[6,7]` is the list `[7,6]`. Here, the axiom $\texttt{rev([])} \twoheadrightarrow \texttt{[]}$ states that the result of reversing an empty list `[]` is again `[]`. The axiom $\texttt{rev(x:xs)} \twoheadrightarrow \texttt{rev(xs) ++ [x]}$ states that a list that begins with an element `x` and continues with a list `xs` can be reversed by first reversing `xs` and then moving `x` to the end.

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\texttt{rev([6,7])}}{\texttt{rev([7]) ++ [6]}}\ \texttt{rev(x:xs)} \twoheadrightarrow \texttt{rev(xs) ++ [x]}}{\texttt{(rev([]) ++ [7]) ++ [6]}}\ \texttt{rev(x:xs)} \twoheadrightarrow \texttt{rev(xs) ++ [x]}}{\texttt{([] ++ [7]) ++ [6]}}\ \texttt{rev([])} \twoheadrightarrow \texttt{[]}}{\texttt{[7] ++ [6]}}\ \texttt{[] ++ xs} \twoheadrightarrow \texttt{xs}}{\texttt{[7,6]}}\ \texttt{[x] ++ xs} \twoheadrightarrow \texttt{x:xs}$$

Definition 8 Let t be an L-term. Then, the *size* of t is defined as the minimum number of leaf nodes in an L-parse tree of t.

The size of t might model the load of t on the working memory.

Example 8 Let `Arith` be as defined in Example 2. Then, the `Arith` terms `f(x+1)`, `f(x) + 2`, and `x + 23` have sizes 1, 3, and 4, respectively. Note the role played by the syntactic category `Chunk` in this case.

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

- L is a language,
- T is an L-theory (declarative memory),
- W is a natural number (working memory capacity),
- S is a natural number (assimilation capacity), and
- D is a natural number (accommodation capacity).

Definition 10 Let L be a language, and let $A = (L, T, W, S, D)$ be an agent. An *A-computation* is a sequence of L-terms $(t_0, \ldots, t_n)$ with

- bounded transitions: $(t_0, \ldots, t_n)$ is a T-computation
- bounded width: no t_i can have a size that exceeds W
- bounded length: $n \leq S$.

As will be further clarified in Definition 14, this definition aims to capture those computations that are within reach of an unassisted human with language L, declarative memory T, working memory capacity W, assimilation capacity S, and accommodation capacity D.

Example 9 Let $L = \texttt{Stream}$, as defined in Example 1 and let T consist of all the L-axioms that are used in the computation of Example 4. Then, $A = (L, T, 8, 10, 6)$ is an agent, and the aforementioned computation is an A-computation.

Observation 1 Let A be an agent. Then, the set of A-computations is finite.

Definition 11 Let L be a language. An *L-item* is a triple (in, out, u) such that *in* and *out* are L-terms, and u is an integer (utility). An *L-induction problem* (IP) is a finite set of L-items.

This definition is slightly more general than the ordinary definition of induction problem with positive and negative examples, which is used in inductive logic programming. The present definition enables us to treat induction problems as more fine-grained optimization problems. It may be helpful to consider utility as rewards (for positive values) and punishments (for negative values). We obtain the ordinary definition of induction problem by assigning utilities of +1 and -1 to the positive and negative examples, respectively.

Definition 12 The *performance* of an agent A on an L-induction problem I is the number

$$\Sigma\{u : (in, out, u) \in I \text{ and } in \to_A out\}.$$

Here, $in \to_A out$ means that there is an A-computation from *in* to *out*. We use the convention that $\Sigma\emptyset = 0$ to ensure that the performance is always defined.

Observation 2 The performance measure is computable.

When all parameters are clear from the context, we will discuss the performance of theories rather than of agents in some cases.

Definition 13 The *size* of an axiom is the sum of the sizes of its left and right terms. The *size* of a theory is the sum of the sizes of its axioms.

Definition 14 The *Occam function* takes as arguments an agent $A = (L, T, W, S, D)$ and an L-induction problem I. The function's value is an L-theory Δ of maximum size D, such that the agent $(L, T + \Delta, W, S, D)$ performs optimally on I in the sense that no other agent $(L, T + \Delta', W, S, D)$, where Δ' is at most of size D, performs strictly better on I. Moreover, the following tiebreakers apply to Δ in the stated priority order:

1. Δ has the minimum size.
2. Δ contains the maximum number of variable occurrences.
3. Δ contains the minimum number of variables.
4. Δ is lexicographically minimal.

If there is no sufficiently small improvement Δ over T, then the Occam function outputs $\emptyset$. Condition 1 aims to capture Occam's razor. Condition 2 states a preference of generality in the sense that variables are preferred over constants. Condition 3 states that variables should be reused whenever possible. Condition 4 ensures that the output is always uniquely defined.

Observation 3 The Occam function is computable.

This follows since there are only finitely many theories and associated agent computations to check.

Results

We have developed a program called OCCAM, which is based on the Occam function and uses an additional strategy for generating axioms by replacing terms by variables. The program first applies several filters to the set of candidate theories and then it evaluates each remaining candidate theory on the relevant induction problem using resource-bounded computations. The program comprises approximately 1,500 lines of code in the functional programming language Haskell.

In this section, we describe two learning processes in which OCCAM is used for learning regularities from a small number of positive and negative examples. The agents considered in this section have the capacity limits $W = 8$, $S = 10$, and $D = 6$. OCCAM finds all the solutions discussed in this section in a matter of minutes. The first learning process concerns English grammar and the challenge is to learn which sequences of words correspond to grammatically correct sentences.

Example 10 To model the first learning situation, let $L =$ `Stream`, as defined in Example 1 and $T = \emptyset$. Also, suppose the IP consists of the items

```
(Alice crawls, OK, 1)        (1)
(Alice, OK, -1).             (2)
```

Then, OCCAM returns the theory Δ consisting of the axiom

```
Alice crawls ↠ OK.           (3)
```

Item (1) is computable in $T + \Delta$ as follows:

$$\frac{\texttt{Alice crawls}}{\texttt{OK}}\ (3)$$

However, item (2) is not computable in $T + \Delta$. In fact, there is no axiom in $T + \Delta$ with a left side that matches `Alice`. A more general way of demonstrating the non-computability of an item in a given theory is to generate all of the computations of this theory and verify that none of them start and end with the relevant terms. Indeed, this is the manner in which OCCAM works. Therefore, the performance of the agent on IP is 1, which is optimal. Intuitively, the first English sentence encountered by the agent was learned by heart.

Example 11 Let us now assume that the agent continues the learning process with T consisting of the axiom (3) from the previous example. Let the IP be given by

```
(Alice plays, OK, 1)         (4)
(Alice, OK, -1).             (5)
```

Then Δ consists of the axiom

```
plays ↠ crawls.              (6)
```

Item (4) is computable in $T + \Delta$ as follows:

$$\frac{\dfrac{\texttt{Alice plays}}{\texttt{Alice crawls}}\ (6)}{\texttt{OK}}\ (3)$$

Intuitively, the agent does not learn the second grammatical sentence that it encounters by heart. Instead, it assumes that replacing `plays` by `crawls` preserves grammatical correctness.

Example 12 We then let T consist of the previously learned axioms (3) and (6). Let the IP be given by

```
(Bob crawls, OK, 1)      (7)
(Alice, OK, -1).         (8)
```

Then, Δ consists of the axiom

```
Bob ↠ Alice.             (9)
```

Item (7) is computable in $T+\Delta$ as follows:

$$\dfrac{\dfrac{\texttt{Bob crawls}}{\texttt{Alice crawls}}\,(9)}{\texttt{OK}}\,(3)$$

Intuitively, the agent draws the conclusion that replacing `Bob` by `Alice` preserves grammatical correctness.

Example 13 We now let T consist of the previously learned axioms (3), (6), and (9). The IP is then given by

```
(Bob plays, OK, 1)       (10)
(Alice, OK, -1).         (11)
```

Then, $\Delta = \emptyset$, and item (10) is computable in $T+\Delta$, as in Example 12. Intuitively, no learning occurred, as the sentence can be analyzed with previously learned knowledge.

Now let us consider a learning process that concerns elementary arithmetic and the challenge of learning a theory of arithmetic from examples.

Example 14 Let $L = \texttt{Arith}$, as defined in Example 2 and $T = \emptyset$. Also, suppose the IP consists of the following items:

```
(5+0, 5, 1)              (12)
(5+1, 5, -1).            (13)
```

Then, OCCAM returns the axiom

```
x+0 ↠ x.                 (14)
```

Thus, OCCAM finds the identity axiom for `+` in this case. Item (12) is computable in $T+\Delta = \{\texttt{x+0} \twoheadrightarrow \texttt{x}\}$ with a one-step computation. However, item (13) is not computable in this theory, as the theory contains no axiom with a left-hand side that matches `5+1`. Thus, the performance of $T+\Delta$ is 1, which is optimal.

OCCAM does not output the axioms `x` $\twoheadrightarrow$ `y`, `x+y` $\twoheadrightarrow$ `x`, or, `x` $\twoheadrightarrow$ `5`, as each of these axioms enables item (13) to be computable.

Example 15 We now let T consist of the previously learned axiom (14) and consider the IP given by

```
(2+3, 3+2, 1)            (15)
(1+3, 3+2, -1).          (16)
```

Then, OCCAM returns the axiom

```
x+y ↠ y+x.               (17)
```

Thus, OCCAM finds the commutativity axiom for `+` in this case. Item (15) is computable in $T+\Delta$ as follows:

$$\dfrac{\texttt{2+3}}{\texttt{3+2}}\,(17)$$

However, item (16) is not computable in $T+\Delta$, as can be seen by inspecting the axioms of $T+\Delta$.

Example 16 Now, the learning process can be continued in a similar manner so that the agent also learns the table entry `2+2` $\twoheadrightarrow$ `4` and the associative law `(x+y)+z` $\twoheadrightarrow$ `x+(y+z)`. Then, the agent is ready to learn ordinary multiplication from examples. In fact, suppose that the IP is given by

```
(g(2,1), 2, 1)           (18)
(g(2,2), 4, 1)           (19)
(g(2,3), 2, -1).         (20)
```

Then, Δ consists of the following axioms:

```
g(x,0) ↠ 0               (21)
g(x,y+1) ↠ g(x,y) + x.   (22)
```

Thus, in this case, OCCAM finds the ordinary recursive definition of multiplication in terms of addition. Item (19) is computable in $T+\Delta$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{g(2,2)}}{\texttt{g(2,1)+2}}\,(22)}{\texttt{(g(2,0)+2)+2}}\,(22)}{\texttt{g(2,0)+(2+2)}}\,\texttt{(x+y)+z} \twoheadrightarrow \texttt{x+(y+z)}}{\texttt{g(2,0)+4}}\,\texttt{2+2} \twoheadrightarrow \texttt{4}}{\texttt{0+4}}\,(21)}{\texttt{4+0}}\,(17)}{4}\,(14)$$

Example 17 Continuing with the theory T that consists of the axioms that the agent has learned thus far, it can now extrapolate number sequences, such as `8, 11, 14`. In fact, let the IP be given by

```
(f(0), 8, 1)             (23)
(f(1), 11, 1)            (24)
(f(2), 14, 1)            (25)
(f(0), 0, -1).           (26)
```

Then, Δ is the following theory:

```
f(0) ↠ 8                 (27)
f(x+1) ↠ f(x) + 3.       (28)
```

Thus, in this case, OCCAM finds a pattern in the number sequence. Item (25) is computable as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{f(2)}}{\texttt{f(1) + 3}}\,(28)}{\texttt{(f(0) + 3) + 3}}\,(28)}{\texttt{f(0) + (3 + 3)}}\,\texttt{(x+y)+z} \twoheadrightarrow \texttt{x+(y+z)}}{\texttt{f(0) + 6}}\,\texttt{3+3} \twoheadrightarrow \texttt{6}}{\texttt{8 + 6}}\,(27)}{14}\,\texttt{8+6} \twoheadrightarrow \texttt{14}$$

We can now use this function to compute the value of `f(3)` and thus obtain the next number of the sequence 17. The same method can be applied to extrapolation and interpolation problems involving arbitrary number sequences. From earlier work, we know that a fixed agent similar to the one considered in this example can perform above the average human level with respect to previously unseen IQ tests (Strannegård, Nizamani, et al., 2013).

Conclusions

We have presented a multi-domain computational model for symbolic reasoning with bounded cognitive resources that supports reasoning by deduction, induction, and abduction.

The model differs from mainstream models that are used in logic, computer science, and cognitive psychology by incorporating a unilateral model of human cognition that serves the purpose of reducing the computational complexity, while keeping performance at the human level or above. Thus we aim to tackle the combinatorial explosion problem that frequently arises in inductive logic programming, automatic theorem proving, and grammar induction.

The model is broad because it can learn entirely new domains of symbolic reasoning by starting with an empty theory. This was shown for simple versions of English grammar and arithmetic. The model also has depth, as it performs above the average human level on several domains, including number sequence extrapolation and tautology identification. We believe that the computational complexity of the model could be improved considerably by adding some natural heuristics. More research is needed to determine the model's generality, scalability, and sensitivity to training data variation.

Acknowledgement

This research was supported by The Swedish Research Council (grant 421-2012-1000).

References

Adler, J. E., & Rips, L. J. (2008). *Reasoning: Studies of Human Inference and its Foundations*. Cambridge University Press.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, N.J.: Lawrence Erlbaum.

Bezem, M., Klop, J. W., & de Vrijer, R. (2003). *Term Rewriting Systems*. Cambridge University Press.

Clark, A., & Lappin, S. (2010). Computational Learning Theory and Language Acquisition. *Philosophy of linguistics*.

Ferri-Ramírez, C., Hernández-Orallo, J., & Ramírez-Quintana, M. J. (2001). Incremental Learning of Functional Logic Programs. In *Functional and Logic Programming*. Springer.

Garcez, A. S. d., & Lamb, L. C. (2011). Cognitive Algorithms and Systems: Reasoning and Knowledge Representation. In V. Cutsuridis, A. Hussain, & J. G. Taylor (Eds.), *Perception-Action Cycle*. Springer New York.

Gobet, F., & Lane, P. (2010). The CHREST Architecture of Cognition: the Role of Perception in General Intelligence. In *Artificial General Intelligence 2010, Lugano, Switzerland*. Atlantis Press.

Katayama, S. (2005). Systematic search for lambda expressions. *Trends in functional programming*, *6*, 111–126.

Kitzelmann, E. (2010). Inductive Programming: A Survey of Program Synthesis Techniques. In *Approaches and Applications of Inductive Programming*. Springer.

Kosslyn, S. M., & Smith, E. E. (2006). *Cognitive Psychology: Mind and Brain*. Upper Saddle River, NJ: Prentice-Hall.

Kühnberger, K.-U., Rudolph, S., & Wang, P. (2013). *Proceedings of the 6th International Conference on Artificial General Intelligence, Beijing, China* (Vol. 7999). Springer.

Laird, J., Newell, A., & Rosenbloom, P. (1987). Soar: An Architecture for General Intelligence. *Artificial Intelligence*, *33*(3), 1–64.

Li, M., & Vitányi, P. M. B. (2009). *An introduction to Kolmogorov complexity and its applications*. Springer.

Muggleton, S., & Chen, J. (2012). Guest editorial: special issue on Inductive Logic Programming (ILP 2011). *Machine Learning*, 1–2.

Olsson, J. R. (1998). The art of writing specifications for the ADATE automatic programming system. In *3rd Annual Conference on Genetic Programming* (pp. 278–283).

Peirce, C. (1958). *Collected Papers of Charles Sanders Peirce*. Belknap Press of Harvard University Press.

Piaget, J. (1937). *La construction du réel chez l'enfant*. Delachaux & Niestlé.

Robinson, J. A., & Voronkov, A. (2001). *Handbook of Automated Reasoning*. Elsevier.

Strannegård, C., Engström, F., Nizamani, A. R., & Rips, L. (2013). Reasoning About Truth in First-Order Logic. *Journal of Logic, Language and Information*, 1–23.

Strannegård, C., Nizamani, A. R., Sjöberg, A., & Engström, F. (2013). Bounded Kolmogorov complexity based on cognitive models. In K. U. Kühnberger, S. Rudolph, & P. Wang (Eds.), *Proceedings of AGI 2013, Beijing, China*. Springer.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., & Goodman, N. D. (2011). How to Grow a Mind: Statistics, Structure, and Abstraction. *Science*, *331*(6022), 1279–1285.

Toms, M., Morris, N., & Ward, D. (1993). Working Memory and Conditional Reasoning. *The Quarterly Journal of Experimental Psychology*, *46*(4), 679–699.

Troelstra, A., & Schwichtenberg, H. (2000). *Basic Proof Theory*. Cambridge University Press.

Veness, J., Ng, K. S., Hutter, M., Uther, W., & Silver, D. (2011). A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, *40*(1), 95–142.

Wang, P. (2007). From NARS to a Thinking Machine. In *Proceedings of the 2007 Conference on Artificial General Intelligence* (pp. 75–93). Amsterdam: IOS Press.

Wang, P., & Goertzel, B. (2012). *Theoretical Foundations of Artificial General Intelligence*. Atlantis Press.