Journal of Artificial General Intelligence 7(1) 104-127, 2016
DOI: 10.1515/jagi-2016-0002

Submitted 2015-09-26
Accepted 2016-07-17

Learning and Reasoning in Unknown Domains

Claes Strannegård CLAES.STRANNEGARD@CHALMERS.SE
Department of Applied Information Technology, Chalmers University of Technology, Sweden and
Department of Philosophy, Linguistics and Theory of Science, University of Gothenburg, Sweden

Abdul Rahim Nizamani ABDULRAHIM.NIZAMANI@GU.SE
Department of Applied Information Technology, University of Gothenburg, Sweden

Jonas Juel JONAS@JUEL.NU

Ulf Persson ULF.PERSSON@CHALMERS.SE
Department of Mathematical Sciences, Chalmers University of Technology, Sweden

Editor: Florin Popescu

Abstract

In the story *Alice in Wonderland*, Alice fell down a rabbit hole and suddenly found herself in a strange world called Wonderland. Alice gradually developed knowledge about Wonderland by observing, learning, and reasoning. In this paper we present the system ALICE IN WONDERLAND that operates analogously. As a theoretical basis of the system, we define several basic concepts of logic in a generalized setting, including the notions of domain, proof, consistency, soundness, completeness, decidability, and compositionality. We also prove some basic theorems about those generalized notions. Then we model Wonderland as an arbitrary symbolic domain and Alice as a cognitive architecture that learns autonomously by observing random streams of facts from Wonderland. Alice is able to reason by means of computations that use bounded cognitive resources. Moreover, Alice develops her belief set by continuously forming, testing, and revising hypotheses. The system can learn a wide class of symbolic domains and challenge average human problem solvers in such domains as propositional logic and elementary arithmetic.
Keywords: autonomous agent, bounded rationality, arbitrary symbolic domain.

1. Introduction

Millennia of evolutionary processes have given animals a remarkable ability to adapt to new environments. Crustaceans can adapt to new environments by using simple forms of learning (Wiese, 2002). Bees can recognize flowers (Clarke et al., 2013) and learn to recognize patterns that are not part of their natural environment, such as letters of the Latin alphabet (Gould, Gould, and others, 1988). Humans are able to adapt to immensely dissimilar environments, including rainforests, deserts, mountains, cities, space stations, and manage to survive there. They can learn any language, any game, any choreography, any musical piece, any script and symbolic domains such as mathematics, logic, and programming. In contrast, artificial systems do not have the same degree of adaptability. Although many artificial systems have been inspired by biological systems, they are mostly designed for specific domains and do not generalize to other domains.

Cognitive architectures are software frameworks that mimic human cognition to varying degrees. These include Soar (Laird, 2012), ACT-R (Anderson and Lebiere, 1998), MicroPsi (Bach, 2012), OpenCog (Goertzel, Pennachin, and Geisweiller, 2014), Sigma (Rosenbloom, 2013), AERA (Steunebrink et al., 2013), and several others. Of particular interest in the present context is the system NARS (Wang and Hammer, 2015), which typically starts with an empty belief-set, receives information from the environment in the form of pairs of terms, and digests this new piece of information by updating its belief set. The system ALICE IN WONDERLAND that will be presented in the present paper shares these properties with NARS, but differs from NARS in many other ways, e.g. by being axiomatic.

Inductive program synthesis and inductive logic programming in particular concerns program synthesis from examples of input-output pairs (Muggleton and Chen, 2012). This is essentially done by compressing the information represented in the examples (Dowe, Hernández-Orallo, and Das, 2011). The analytic approach to inductive program synthesis uses the examples together with, e.g. anti-unification and recursive relation learning to produce the programs (Kitzelmann, 2010). The generate-and-test approach uses the examples for testing purposes only (Katayama, 2015). The computational complexity of both these methods is a major obstacle to the use of inductive program synthesis on a larger scale (Kitzelmann, 2010). In this paper we try to tackle this complexity problem by using bounded cognitive resources for reducing the size of the search space (Strannegård et al., 2013).

Turing (Turing, 1950, p.17) proposed a developmental psychology approach for building intelligent machines:

> "Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's? If this were then subjected to an appropriate course of education one would obtain the adult brain."

This gave rise to the idea of developing systems. How, then, do children learn and acquire new knowledge? According to Piaget, children adapt to new information in one of two ways: by *assimilation*, in which new information fits into existing knowledge structures; and *accommodation*, in which new information causes new knowledge structures to form or old ones to be modified (Piaget, 1937). Popper provides an evolutionary perspective on knowledge acquisition (Popper, 1972, p.261):

> "The growth of our knowledge is the result of a process closely resembling what Darwin called 'natural selection'; that is, the natural selection of hypotheses: our knowledge consists, at every moment, of those hypotheses which have shown their (comparative) fitness by surviving so far in their struggle for existence, a competitive struggle which eliminates those hypotheses which are unfit."

In this paper we will present the system ALICE IN WONDERLAND that was inspired by the story about Alice in Wonderland (Carroll, 2000). In this story, Alice fell into a rabbit hole and suddenly found herself in Wonderland, a world that was completely strange to her. Over time she gathered experiences from Wonderland and used observations along with inductive and deductive reasoning to gradually evolve a set of beliefs. This set of beliefs enabled her to become better and better at making predictions about Wonderland. We model Alice as a simple cognitive architecture that includes an evolving belief set together with mechanisms for observing, learning and reasoning.

A general notion of domain is used to model Wonderland, which is described as a finite or infinite set of facts. Alice makes observations about Wonderland via a random stream of such facts.

All cognitive resources of Alice, including her long-term and working memory capacities, are strictly bounded. Thus we model some cognitive limitations of Alice and at the same time we will restrict the reasoning capacity and thereby the computational complexity of the system. The system builds on theories borrowed from developmental psychology (Von Glasersfeld, 1995), along with bounded rationality (Simon, 1982), proof theory (Negri, Von Plato, and Ranta, 2008), belief revision (Hansson et al., 2001), and inductive program synthesis (Kitzelmann, 2010).

ALICE IN WONDERLAND was developed as a modification of a traditional approach to reasoning that is common in mathematics and formal verification. This approach is logic-based and uses a vocabulary that is designed by humans, formulas that are fixed, axioms that are designed by humans, axiom sets that remain constant over time, and proof rules that are fixed. Moreover, the proof complexity is only considered in the limit or not at all, while proof comprehensibility is never an issue. We will modify this approach in order to make it more dynamic. In particular we will relax the definition of formulas and add the ability to learn and reason inductively. We will also introduce a simple model of bounded cognitive resources, which has the side-effect that the system automatically becomes decidable. We will also define several basic concepts of logic in a generalized setting, e.g. the notions of domain, proof, consistency, soundness, completeness, decidability, and compositionality. We will also prove some basic theorems about those generalized concepts.

Section 2 introduces the basic terminology. Section 3 presents the notion of abstract domain, which is used for modeling Wonderland. Sections 4–9 describe how Alice represents beliefs about Wonderland, draws conclusions, solves open and closed problems, interacts with the environment, evaluates hypotheses, and updates her beliefs, respectively. Results are presented in section 10 and section 11 concludes the paper. A table of the symbols used in the paper can be found in Appendix A and the main algorithms of ALICE IN WONDERLAND are given in Appendix B.

2. Preliminaries

In this section we will introduce some basic notation.

Definition 1 (symbol) *A* symbol *is a Unicode character, other than the characters* (,)*, and* □*, which will be used later for special purposes.*

The choice of Unicode as the theoretical symbol base was made because we wanted to work with languages with special character sets such as Sindhi and Swedish. While Unicode is a huge character set, each domain will typically only require a small subset of Unicode and Alice will only consider terms over the vocabulary of the relevant domain.

Definition 2 (variable) *A* variable *is a symbol belonging to the set* {x, y, z}.

Definition 3 (vocabulary) *A* vocabulary *is a set of finite symbol strings.*

Example 1 *Here are some vocabularies that will be used in our running examples in the following:*

Arithmetic vocabulary: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 *(digits),* + *(addition),* ⋆ *(multiplication), # (concatenation).*

Propositional vocabulary: `p`, `q`, `r` *(propositional constants),* $\top$ *(truth),* $\bot$ *(falsity),* $\neg$ *(negation),* $\wedge$ *(conjunction),* $\vee$ *(disjunction),* $\rightarrow$ *(implication).*

English vocabulary: `_` *(space),* `Alice`, `Stella`, `runs`, `plays`, `fast`, *along with all other words appearing in a given English corpus.*

Definition 4 (term) *Let V be a vocabulary. A V-*term *is a finite ordered labeled tree. Each node is labeled with either an element of V or a variable. A V-term is* open *if it has a node that is labeled with a variable and* closed *otherwise.*

We follow common practice and write terms as strings, using parentheses and standard conventions for disambiguation.

Example 2 *Here are some examples of terms:*

Arithmetic terms: `2*(3+4)`, 1#2, `x*0`

Propositional terms: $\neg\bot$, `x`$\wedge$`y`, `x`$\vee\top$

English terms: `Alice_plays`, `Stella_(runs_fast)`, `OK`

Here the operator `#` is used for concatenating elements, regardless of vocabulary: e.g., the number `12` is represented by the term `1#2`. This is achieved by not counting `#` when determining the size of a term. Usually we will omit both `#` and `_` to facilitate reading. Note that we have imposed no type or arity restrictions on the terms. For instance, a labeled tree such as `1(*)` also qualifies as a V_A-term.

Other examples of terms of this kind could be formulas of first-order or second-order logic.

Definition 5 (language) *A* language *over a vocabulary V is a set of V-terms.*

Languages of quantifier-free arithmetic, propositional logic, and a certain context-free fragment of English can be defined in a standard way using recursive definitions. We assume that this has been done, resulting in the languages L_A, L_P, and L_E, respectively.

Example 3 *Here are some examples of elements of the languages L_A, L_P, and L_E, respectively:*

Arithmetic language: `2*(3+4)`, 1#2, `x*0`

Propositional language: $\neg\bot$, `x`$\wedge$`y`, `x`$\vee\top$

English language: `Alice_plays`, `Stella_(runs_fast)`.

Definition 6 (substitution) *A* substitution *is a partial function σ that assigns terms or labels to variables. The substitution σ can be applied to terms by replacing variables at leaf nodes by terms and variables at non-leaves by labels according to σ.*

Example 4 $\emptyset$, $\{$`x`$=$`Alice`$\}$, *and* $\{$`x`$=2,$`y`$=3\}$ *are substitutions. If* $\sigma=\{$`x`$=2,$`y`$=3\}$, *then* σ(`x+y`)$=$`2+3`. *Also, if* $\sigma'=\{$`x`$=$`raven`$\}$, *then* σ'(`x(hugin)`)$=$`raven(hugin)`.

Definition 7 (context) *A* context *is a term t containing exactly one occurrence of the symbol* $\Box$, *which labels a leaf of t.*

Example 5 $\Box+5$, $\top\vee\Box$, *and* $\Box$ `runs` *are contexts.*

If c is a context and t is a term, then $c(t)$ is the result of replacing the unique occurrence of $\Box$ in c by t. If $c=\Box$, then $c(t)=t$.

3. Wonderland

In this section we introduce our model of Wonderland, the environment surrounding Alice. We will also generalize some semantic notions from propositional logic, namely consequence, equivalence, and compositionality. The new concepts are presented together with plenty of examples to make the large amount of definitions easier to grasp. Many theorems follow more or less directly from definitions and in those cases their proofs are omitted.

Definition 8 (fact) *Let L be a language. An L-*fact *is an expression of the form $t \vdash t'$ or $t \nvdash t'$, where t and t' are closed terms of L.*

Example 6 *Here are some examples of facts:*

Arithmetic facts: `6+6` $\vdash$ `12`, `0` $\nvdash$ `1`

Propositional facts: $\neg\top \vdash \bot$, $\top \wedge \bot \nvdash \top$, `runs` $\nvdash$ `OK`

English facts: `Stella runs fast` $\vdash$ `OK`.

Definition 9 (domain) *Let L be a language. An L-*domain *is a set of L-facts D such that for all t and t', $t \vdash t'$ and $t \nvdash t'$ are not both elements of D.*

Example 7 *Here are examples of three different domains defined over L_A, L_P, and L_E, respectively:*

Arithmetic domain: $\{t \vdash t' : t = t'\} \cup \{t \nvdash t' : t \neq t'\}$

Propositional domain: $\{t \vdash t' : t$ *is a logical consequence of* $t'\}$

English domain: $\{t \vdash$ `OK` $: t \in L_E\} \cup \{t \nvdash$ `OK` $: t \notin L_E\}$

Definition 10 (domain language) *Let D be a domain. Then we define $L(D) = \{t : t \vdash t' \in D$ for some $t'\} \cup \{t' : t \vdash t' \in D$ for some $t\}$ and say that $L(D)$ is the* domain language *of D.*

We will work with trees of a particular form.

Definition 11 (left shuffled tree) *A tree is* left shuffled *if and only if the subtree generated by any of its nodes has the property that only its leftmost subtree has size larger than one.*

Example 8 *Here are some examples illustrating this concept (with trees written as strings):*

The following is left shuffled: `(1 + 2) + 3`.

The following is not left shuffled: `1 + (2 + 3)`.

Definition 12 (constants) *Let D be a domain. Then we define the set of* constants *of D, $C(D) = \{s : s$ is a left shuffled subtree of $t \in L(D)$ that consists of only leaves and `#`$\}$.*

Example 9 *Here are some examples of constants:*

Arithmetic constants: `6`, `1#2`, `(1#2)#3`

Propositional constants: $\top$, $\bot$

English constants: `Alice`, `runs`, `OK`.

Definition 13 (reflexivity, symmetric and transitive) *We say that the domain D is* reflexive / symmetric / transitive *if the relation $\{(t,t') : t \vdash t' \in D\}$ is reflexive / symmetric / transitive.*

We now proceed to extend the $\vdash$ relation to open terms.

Definition 14 (consequence) *Let $t \models_D t'$ iff $\sigma(t) \vdash \sigma(t') \in D$ holds for every substitution σ such that $\sigma(t) \in L(D)$. We call $\models_D$ the* consequence *relation of D.*

Theorem 15 *The following equivalences hold:*

1. *D is reflexive iff $\models_D$ is reflexive*
2. *D is symmetric iff $\models_D$ is symmetric*
3. *D is transitive iff $\models_D$ is transitive.*

Definition 16 (similarity) *Let $t \sim_D t'$ iff $t \models_D t'$ and $t' \models_D t$. We call $\sim_D$ the* similarity *relation of D.*

Definition 17 (equivalence) *Let $t \equiv_D t'$ iff $c(t) \sim_D c(t')$ holds for all contexts c such that $c(t), c(t') \in L(D)$. We call $\equiv_D$ the* equivalence *relation of D.*

Theorem 18 *The following holds:*

1. *$\equiv_D$ is symmetric*
2. *D is reflexive iff $\equiv_D$ is reflexive*
3. *D is transitive iff $\equiv_D$ is transitive.*

Theorem 19 *Suppose $t \equiv_D t'$. Then $\sigma(c(t)) \equiv_D \sigma(c(t'))$ for all σ and c.*

Definition 20 (compositionality) *The domain D is* compositional *if for all t and t', $t \sim_D t'$ implies $t \equiv_D t'$.*

Example 10 *A, P, and E are compositional. Thus:*

- `1+2` $\sim_A$ `3` *implies* `(1+2)*9` $\sim_A$ `3*9`
- `True`$\vee$`False` $\sim_P$ `True` *implies* `(True`$\vee$`False)`$\wedge$`False` $\sim_P$ `True`$\wedge$`False`
- `Stella runs` $\sim_E$ `Alice runs` *implies* `Stella runs fast` $\sim_G$ `Alice runs fast`

Definition 21 (term size) *Given term t, size(t) is the number of nodes of t, excluding nodes labeled with the concatenation symbol #.*

Theorem 22 *For some D, $\models_D$ and $\equiv_D$ are undecidable relations.*

Proof Let Ω be any undecidable set of natural numbers. Also let D be the following domain over the language of arithmetic L_A

$$\{(t \vdash t : t \text{ represents a number in } \Omega\}.$$

Then $\models_D$ is undecidable (and reflexive and transitive). Moreover, $\models_D = \equiv_D$. ■

4. How Alice represents beliefs

In this section we will begin to introduce our model of Alice. Alice is only in contact with Wonderland via a stream of observations. Thus she has no other information channels and in particular she has no *a priori* knowledge about the language of the domain.

Definition 23 (stream) *Let D be a domain. A D-*stream *is a sequence of facts $F_0, F_1, \ldots$, where each $F_i \in D$.*

Alice will store her beliefs about Wonderland in the form of rules:

Definition 24 (rule) *A* rule *is an expression of the form $t \rhd t'$ (shallow rule), $t \blacktriangleright t'$ (deep rule), or $t \not\rhd t'$, where t and t' are terms (either open or closed).*

Example 11 *The following are examples of rules: $0 \not\rhd 1$, $2*2 \blacktriangleright 4$, $\texttt{Alice plays} \rhd \texttt{OK}$, and $x \vee y \rhd x$.*

The syntactic relations $\rhd$, $\blacktriangleright$, and $\not\rhd$ will be used for describing the semantic relations $\models$, $\equiv$, and $\not\models$, respectively.

Definition 25 (purity) *The rule $t \rhd t'$ / $t \blacktriangleright t'$ / $t \not\rhd t'$ is* pure *if all variables of t' appear in t.*

Example 12 *The rules $x*0 \rhd 0$, $x \vee \top \blacktriangleright \top$, and $\texttt{plays} \rhd \texttt{OK}$ are pure, whereas $0 \rhd x*0$ and $\top \blacktriangleright x \vee \top$ are not.*

Purity will be helpful for us in two ways. First, it will ensure that types are preserved. For instance, if $t \rhd t'$ is a pure rule of arithmetic and t is a term of arithmetic, then t' will also be a term of arithmetic. Secondly, purity will ensure decidability, as we will see later. Next we shall see how Alice represents her set of beliefs:

Definition 26 (theory) *A* theory *is a finite set of pure rules.*

Example 13 *Here are examples of theories:*

Arithmetic theory: $\{2*2 \blacktriangleright 4, x*0 \blacktriangleright 0, x+y \blacktriangleright y+x\}$

Propositional theory: $\{x \vee \top \blacktriangleright \top, x \rightarrow y \blacktriangleright \neg x \vee y, x \vee y \rhd x\}$

English theory: $\{\texttt{Alice plays} \blacktriangleright \texttt{OK}, \texttt{Alice} \blacktriangleright \texttt{Stella}, \texttt{runs} \not\rhd \texttt{OK}\}$.

Definition 27 (deep rules) *Let $deep(T)$ be the set of rules of T that have the form $t \blacktriangleright t'$.*

Definition 28 (closed rules) *Let $cl(T)$ the set of rules of T that consist of closed terms only.*

5. How Alice draws conclusions

In this section we show how Alice draws conclusions. We begin by presenting unbounded computations and then proceed to introduce Alice's bounded computations that are restricted by limitations on her cognitive resources.

$$\dfrac{\dfrac{\dfrac{\dfrac{\texttt{(2+4)*(6+1)}}{\texttt{6*(6+1)}}\ \texttt{2+4} \blacktriangleright \texttt{6}}{\texttt{6*7}}\ \texttt{6+1} \blacktriangleright \texttt{7}}{\texttt{42}}\ \texttt{6*7} \blacktriangleright \texttt{42}$$

$$\dfrac{\dfrac{\texttt{0*12}}{\texttt{12*0}}\ \texttt{x*y} \blacktriangleright \texttt{y*x}}{\texttt{0}}\ \texttt{x*0} \blacktriangleright \texttt{0}$$

Figure 1: These computations can be interpreted as arithmetic computations with rules that preserve equality.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\texttt{(p} \rightarrow \texttt{q)} \vee \texttt{p}}{(\neg\ \texttt{p} \vee \texttt{q)} \vee \texttt{p}}\ \texttt{x} \rightarrow \texttt{y} \blacktriangleright \neg\texttt{x} \vee \texttt{y}}{\texttt{(q} \vee \neg\ \texttt{p)} \vee \texttt{p}}\ \texttt{x} \vee \texttt{y} \blacktriangleright \texttt{y} \vee \texttt{x}}{\texttt{q} \vee (\neg\ \texttt{p} \vee \texttt{p})}\ (\texttt{x} \vee \texttt{y}) \vee \texttt{z} \blacktriangleright \texttt{x} \vee (\texttt{y} \vee \texttt{z})}{\texttt{q} \vee \top}\ \neg\texttt{x} \vee \texttt{x} \blacktriangleright \top}{\top}\ \texttt{x} \vee \top \blacktriangleright \top$$

$$\dfrac{\texttt{black(Hugin)}}{\texttt{raven(Hugin)}}\ \texttt{black(x)} \rhd \texttt{raven(x)}$$

Figure 2: These computations can be interpreted as logic computations with rules that preserve or increase logical strength: i.e., goal-driven proofs. To prove $\texttt{(p} \rightarrow \texttt{q)} \vee \texttt{p}$, it is sufficient to prove $\top$; to prove `black(Hugin)`, it is sufficient to prove `raven(Hugin)`

5.1 Unbounded computations

Definition 29 (computation) *Suppose T is a theory. A T-*computation *is a sequence of closed terms $(t_1, \ldots, t_n)$ such that for all k such that $0 < k < n$,*

Shallow step. *$t_k = \sigma(t)$ and $t_{k+1} = \sigma(t')$, for some σ and $t \rhd t' \in T$, or*

Deep step. *$t_k = c(\sigma(t))$ and $t_{k+1} = c(\sigma(t'))$, for some c, σ, and $t \blacktriangleright t' \in T$.*

Definition 30 (computability) *We write $t \vdash_T t'$ if there is a T-computation from t to t'.*

Examples of computations – written vertically and annotated with rules – are given in Figures 1–4.

The following two properties link together theories and domains:

Definition 31 (completeness) *A theory T is* complete *for a domain D if for all $t \vdash t' \in D$ we have $t \vdash_T t'$.*

Definition 32 (soundness) *The theory T is D-*sound *if*

$$\dfrac{\dfrac{\dfrac{\texttt{Stella plays}}{\texttt{Stella crawls}}\ \texttt{plays} \blacktriangleright \texttt{crawls}}{\texttt{Alice crawls}}\ \texttt{Stella} \blacktriangleright \texttt{Alice}}{\texttt{OK}}\ \texttt{Alice crawls} \blacktriangleright \texttt{OK}$$

Figure 3: These computations can be interpreted as English computations: specifically, derivations in formal grammar with rules that preserve grammatical equivalence. The derivation shows that `Stella plays` is a grammatically correct sentence of English.

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

Figure 4: These computations can be interpreted as Haskell computations with rules that preserve equality. The derivation shows how the function `rev` reverses the list `[6,7]`.

- $t \blacktriangleright t' \in T$ *implies* $t \sim_D t'$
- $t \rhd t' \in T$ *implies* $t \models_D t'$
- $t \not\rhd t' \in T$ *implies* $t \not\models_D t'$

Theorem 33 *Suppose T is D-sound and D is reflexive, transitive, and compositional. Then $t \vdash_{deep(T)} t'$ implies $t \equiv_D t'$.*

Proof Suppose T is D-sound. Also suppose $t \vdash_{deep(T)} t'$. We show that $t \equiv_D t'$ by induction on the computation length, defined as the number of computation steps.

Base case (length 0): Then the computation consists of one line. Thus $t = t'$ and hence $t \equiv_D t'$ by the reflexivity of D and Theorem 18.

Induction case (length >0): Suppose the last step of the computation is from t'' to t'. By induction we have $t \equiv_D t''$. We show that $t'' \equiv_D t'$. This is enough since $\equiv_D$ is transitive by Theorem 18 and the assumption that D is transitive.

Suppose the step of the computation from t'' to t' is a deep step. Then $t'' = \sigma(c(s''))$ and $t' = \sigma(c(s'))$, for some $s'' \blacktriangleright s' \in T$. Since T is D-sound, we also have $s'' \sim_D s'$, so compositionality yields $s'' \equiv_D s'$. Hence by Theorem 19 we have $\sigma(c(s'')) \equiv_D \sigma(c(s'))$, in other words $t'' \equiv_D t'$. ■

Theorem 34 *Suppose T is D-sound and D is reflexive, transitive, and compositional. Then $t \vdash_T t'$ implies $t \models_D t'$.*

Proof Suppose T is D-sound. Also suppose $t \vdash_T t'$. We show that $t \models_D t'$ by induction on the computation length.

Base case (length 0): Then the computation consists of one line. Thus $t = t'$ and hence $t \models_D t'$ by the reflexivity of D and Theorem 15.

Induction case (length >0): Suppose the last step of the computation is from t'' to t'. By induction we have $t \models_D t''$. We show that $t'' \models_D t'$. This is enough since $\models_D$ is transitive by Theorem 15 and the assumption that D is transitive.

Suppose the last step of the computation from t'' to t' is a shallow step. Thus $t'' = \sigma(s'')$ and $t' = \sigma(s')$, for some $s'' \rhd s' \in T$. Since T is D-sound, we also have $s'' \models_D s'$. Hence we have $\sigma(s'') \models_D \sigma(s')$, in other words $t'' \models_D t'$. If the last step of the computation from t'' to t' is a deep step, the claim follows by Theorem 33. ■

5.2 Bounded computations

Definition 35 (bounded resources) *The following arbitrarily defined constants determine the boundaries of Alice's cognitive resources.*

- *Long-term memory:* $LTM_{Alice} = 200$ *(rules)*
- *Working memory:* $WM_{Alice} = 8$ *(nodes)*
- *Computation depth:* $Depth_{Alice} = 10$ *(steps).*

Definition 36 (bounded computation) *A* bounded T-computation *is a T-computation* $(t_1, \ldots, t_n)$ *with*

- *bounded length:* $n \leq Depth_{Alice}$
- *bounded width:* $size(t_i) \leq WM_{Alice}$*, for each* $1 \leq i \leq n$.

Intuitively, all bounded computations fit into a frame of height $Depth_{Alice}$ and width WM_{Alice}. The computations in Figures 1–4 are all bounded.

Definition 37 (bounded computability) *We write $t \vdash_T^* t'$ if there is a bounded T-computation from t to t'.*

Theorem 38 *The relation $\vdash_T^*$ is decidable.*

Proof All theories are finite by definition. Given the purity condition on rules, only finitely many bounded T-computations beginning with any given closed term t are possible. The bounded T-computations starting with t form a finitely branching tree where each branch has a maximum length. Hence, the tree is finite and $\vdash_T^*$ is decidable. ■

6. How Alice solves problems

In this section we show how Alice solves problems of two types: closed and open.

Definition 39 (closed problem) *A* closed problem *is a closed term t. A* solution *to closed problem t with respect to T is a term t' of minimum size such that $t \vdash_T^* t'$.*

This definition reflects our view that many computational problems can be regarded as terms that asks for solutions in the form of compressed versions of the problems. This sometimes corresponds to the challenge of turning something less familiar into something more familiar. Several examples of closed problems and solutions are given in Table 1.

Before proceeding to open problems we need to introduce an auxiliary notion:

Definition 40 (adequacy) *The* adequacy *of theory T for (describing) the non-empty theory T' is the number*

$$Adequacy(T, T') = \frac{card(\{t \rhd t' \in T' : t \vdash_T^* t'\} \cup \{t \not\rhd t' \in T' : t \not\vdash_T^* t'\})}{card(T')}.$$

Closed problem	Solution
`13+4`	`17`
$(\mathtt{p} \rightarrow \mathtt{q}) \vee \mathtt{p}$	$\top$
`Stella crawls`	`OK`
`f(2)`	`9`

Table 1: Closed problems with examples of solutions.

The notion of adequacy can be seen as a measure of how well a theory T can approximate a theory T'. More precisely the numerator is the number of rules of T' that can be computed using T.

Example 14 *Given* $T' = \{2 * 0 \rhd 0, 3 * 0 \rhd 0, 1 \not\rhd 0\}$*, the following holds:*

- $Adequacy(\{2 * 0 \blacktriangleright 0\}, T') = 2/3$.
- $Adequacy(\{x \blacktriangleright 0\}, T') = 2/3$.
- $Adequacy(\{x * 0 \blacktriangleright 0\}, T') = 1$.

Definition 41 (open problem) *An* open problem *is a non-empty theory with rules of the form* $t \rhd t'$ *and* $t \not\rhd t'$*. A* solution *to open problem* P *with respect to* T *is a substitution* σ *such that* $Adequacy(T, \sigma(P)) = 1$.

Examples of open problems and their solutions are given in Table 2.

7. How Alice interacts

The system ALICE IN WONDERLAND, which consists of around 5,000 lines of Haskell code, models both Wonderland and Alice. Wonderland is modeled as an arbitrary domain D and Alice's belief set at time n is modeled by a theory T_n. Alice starts with theory T_0, which is empty by default. At any time n, the system can be either in *learning mode* or in *inquiry mode*, as determined by the human operator:

Learning mode: Alice receives fact $F_n \in D$. Alice learns from F_n and updates her theory to T_{n+1}.

Inquiry mode: Alice receives an open or closed problem, P_n. Alice outputs a solution to P_n or reports failure and puts $T_{n+1} = T_n$.

In learning mode, F_n could come from any source, e.g. sensors, a text file, or human entry. Moreover, for purposes of illustration, predefined streams can be chosen from a dropdown menu. In inquiry mode, P_n, which can be an open or closed problem, is entered by the human user. Figure 5 shows a screenshot of the system in operation.

8. How Alice analyzes theories

In this section we will present the analytical tools that Alice uses to update her set of beliefs about Wonderland. Alice learns about Wonderland D from experience and gradually develops a sequence

Open problem	**Solution**
x+4 $\triangleright$ 9	x=5
x+y $\triangleright$ 2	x=0, y=2
(x-1)*(x-2) $\triangleright$ 0	x=1
x runs $\triangleright$ OK	x=Alice
even(x) $\triangleright$ True	x=0
under(x,foot)$\triangleright$ True	x=sole
capital(x,Pakistan)$\triangleright$ True	x=Islamabad
x $\vee$ False $\triangleright$ True	x=True
x(Hugin) $\triangleright$ True	x=raven
x(Hugin) $\triangleright$ raven(Hugin)	x=black
x $\triangleright$ cough, x $\triangleright$ sneeze	x=cold
x $\triangleright$ cough, x $\triangleright$ sneeze, x $\not\triangleright$ cold	x=flu
x+y $\triangleright$ 2, x*y $\triangleright$ 1	x=1, y=1
(x-1)*(x-2) $\triangleright$ 0, x $\not\triangleright$ 1	x=2
x+y $\triangleright$ 2, x*y $\triangleright$ 1, x $\not\triangleright$ 0	x=1, y=1
x(cat,mouse)$\triangleright$ True	x=chases
x(sun,planet)$\triangleright$ True, x(nucleus,electron)$\triangleright$ True	x=circles
x(palm,hand) $\triangleright$ True, x(y,foot) $\triangleright$ True	x=under, y=sole

Table 2: Open problems with examples of solutions.

of theories T_n. Her goal is to eventually understand Wonderland in the sense that for big enough n, $t \models_D t'$ iff $t \vdash_{T_n} t'$. In general, if she only puts closed rules into her theory, then it will not be complete. For instance, in arithmetic the theory would not be able to add 1 to a sufficiently large number. Thus she is forced to use open rules as well. But as soon as she puts an open rule into her theory, she runs the risk that the rule is unsound. In fact, a counterexample may appear at some later stage. Therefore the best Alice can do is use decidable criteria for selecting open rules that *appear* to be sound. These criteria will necessarily be imperfect, however. Closed rules, on the other hand, are easier to deal with. In fact, Alice will only put a closed rule into her theory if she knows that it is sound. The following notion will be used to ensure controlled transitions from T_n to T_{n+1}.

Definition 42 (successor) *Theory T' is a* successor *of T if $T' - T$ contains at most one rule.*

Example 15 *Any subset of $T \cup \{t \blacktriangleright t'\}$, $T \cup \{t \triangleright t'\}$ or $T \cup \{t \not\triangleright t'\}$ is a successor of T.*

Alice responds to the incoming stream $F_0, F_1, \ldots$ by forming the sequence of theories $T_0, T_1, \ldots$. Theory T_{n+1} is selected as a successor of T_n that contains at most LTM_{Alice} rules and fulfills a number of additional requirements. To be able to describe those requirements, we now proceed to introduce some further notions.

8.1 Local convergence

We start with the notion of local convergence, which Alice uses as an (insufficient) indicator of soundness. First we need to define a couple of auxiliary notions.

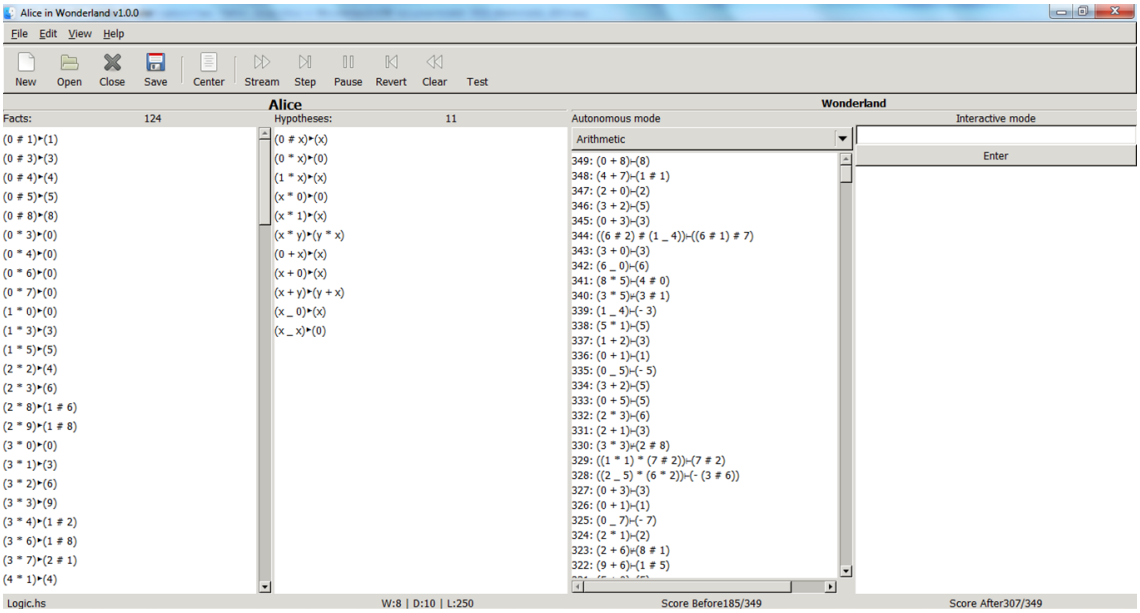


Figure 5: Screenshot of ALICE IN WONDERLAND. The system has been running in learning mode, processing 349 arithmetic facts in the course of around 30 seconds (on a standard laptop). From left, the first and second panels represent Alice's beliefs in the form of closed and open rules, respectively. Alice's current theory consists of 124 closed rules, including `2*3 = 6`, and 11 open rules, including `x*0 = 0`. The third panel shows the fact stream, the fourth panel solutions to problems entered by the human operator.

Definition 43 (T-context) *A* T-context *is a context obtained from a term* t *appearing in a rule of* T *by replacing an occurrence of a subterm of* t *with* $\Box$.

Example 16 *Suppose* `Alice runs` $\blacktriangleright$ `OK` $\in T$. *Then* $\Box$ `runs` *is a* T*-context.*

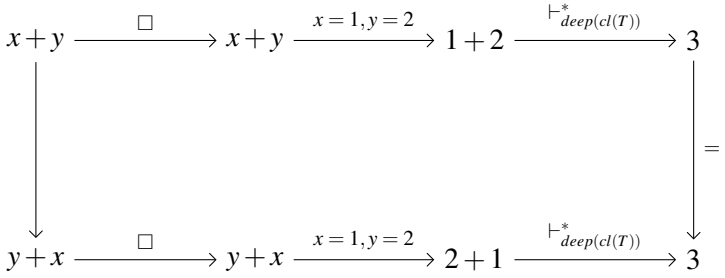
Definition 44 (T-substitution) *A* T-substitution *is a substitution that assigns closed terms that appears in the rules of* T *to variables.*

Theorem 45 *There are only finitely many* T*-contexts and* T*-substitutions.*

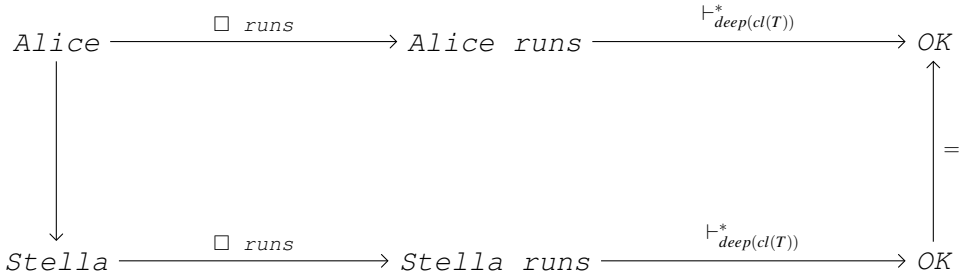
Definition 46 (local convergence) *The rule* $t \blacktriangleright t'$ *is* locally T*-convergent via the* T*-context* c *and the* T*-substitution* σ *if there is a term* s *such that the following holds:*

$$\begin{array}{ccccccc} t & \xrightarrow{c} & c(t) & \xrightarrow{\sigma} & \sigma(c(t)) & \xrightarrow{\vdash^*_{deep(cl(T))}} & s \\ \downarrow & & & & & & \downarrow = \\ t' & \xrightarrow{c} & c(t') & \xrightarrow{\sigma} & \sigma(c(t')) & \xrightarrow{\vdash^*_{deep(cl(T))}} & s' \end{array}$$

Example 17 *The rule $x+y \blacktriangleright y+x$ is locally T-convergent in the following situation:*



Example 18 *The rule* `Alice` $\blacktriangleright$ `Stella` *is locally T-convergent in the following situation:*



Now we come to a central theorem of our approach:

Theorem 47 *Local convergence is decidable.*

Proof There are only finitely many T-contexts and T-substitutions. Also, bounded computability is decidable. ■

The following theorem shows that for certain domains the notion of local convergence implies soundness for closed rules. Thus local convergence can be used both for certifying soundness of encountered closed rules and for indicating soundness of open rules. Since local convergence is decidable we have an effective measure: the more locally convergent situations Alice discovers, the more her confidence increases in the corresponding rules.

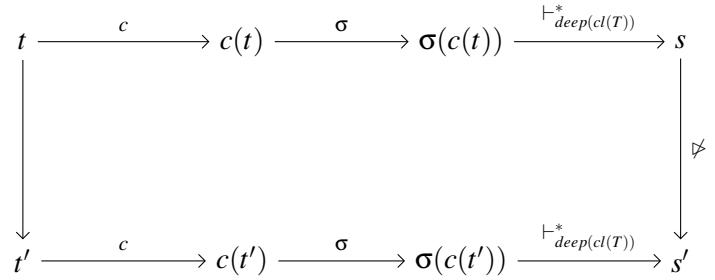
Theorem 48 *Suppose D is reflexive and transitive, compositional and $cl(T)$ is D-sound. Also suppose $t \blacktriangleright t'$ is closed and locally $cl(T)$-convergent. Then $t' \equiv_D t$ and thus $t \blacktriangleright t'$ is sound.*

Proof Suppose $t \blacktriangleright t'$ is closed and locally T-convergent. Then we have $t \vdash_{deep(cl(T))} s$ and $t' \vdash_{deep(cl(T))} s$. Hence by Theorem 33 we get $t \equiv_D s$ and $t' \equiv_D s$. Then transitivity of $\models_D$ and symmetry of $\equiv_D$ yields $t' \equiv_D t$. ■

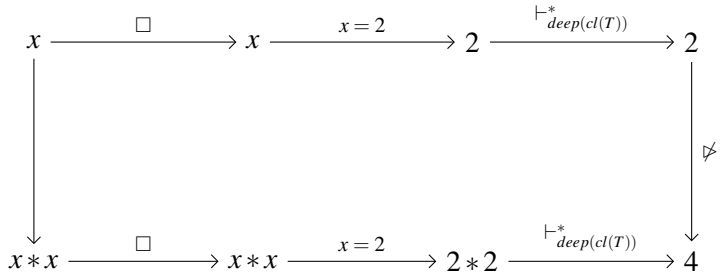
8.2 Local divergence

Next, let us introduce the notion of local divergence, which Alice uses as a (sufficient) indicator of unsoundness.

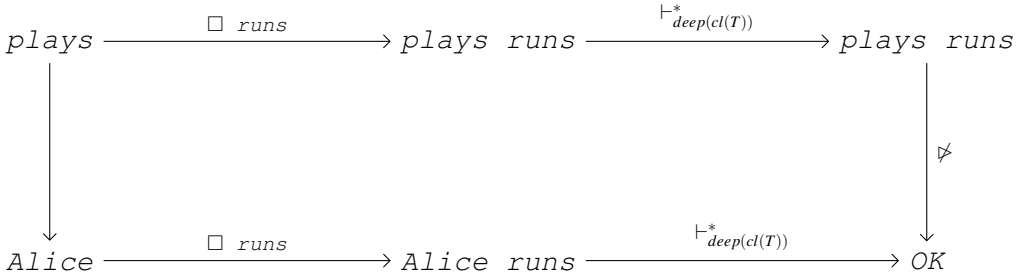
Definition 49 (Local divergence) *The rule $t \blacktriangleright t'$ is* locally T-divergent *if there are a T-context c, a T-substitution σ, and a rule $s \not\blacktriangleright s' \in T$ such that the following relations hold:*



Example 19 *The rule $x \blacktriangleright x * x$ might be locally T-convergent when $x = 0$ and $x = 1$, but locally T-divergent when $x = 2$:*



Example 20 *The rule* `plays` $\blacktriangleright$ `Alice` *is locally T-divergent in the following situation:*



Theorem 50 *Local T-divergence is decidable.*

Proof There are only finitely many T-contexts and T-substitutions. Also, bounded computability is decidable. ■

Theorem 51 *Suppose D is a reflexive, transitive, and compositional domain. Also suppose that $cl(T)$ is D-sound. Then if the rule $t \blacktriangleright t'$ is locally $cl(T)$-divergent, then it is not D-sound.*

Proof We will prove this by contradiction. Suppose $t \blacktriangleright t'$ is D-sound. Then $t \equiv_D t'$. From the local $cl(T)$-divergence we also have $\sigma(c(t)) \vdash^*_{deep(cl(T))} s$ and $\sigma(c(t')) \vdash^*_{deep(cl(T))} s'$ for some c, σ and $s \not\triangleright s' \in cl(T)$. By Theorem 33 this implies $\sigma(c(t)) \equiv_D s$ and $\sigma(c(t')) \equiv_D s'$. Now by Theorem 19 we also have $\sigma(c(t)) \equiv_D \sigma(c(t'))$ and by transitivity it follows that $s \equiv_D s'$. On the other hand we have $s \not\triangleright s' \in cl(T)$. Again by the D-soundness of $cl(T)$ $s \not\equiv_D s'$. Contradiction. ■

The two last theorems show that local divergence can be used for showing unsoundness of open rules.

8.3 Strength

Among the candidate rules that appear to be sound, Alice has a preference for those that are relatively strong. The following measure counts how many rules of T would be subsumed if the candidate rule R would be added.

Definition 52 (strength) *The* strength *of a candidate rule R w.r.t. the theory T is defined as follows:*

$$S(R,T) = card(\{t \rhd t' \in T : t \vdash^*_{T+R-t\rhd t'} t'\}) \cup card(\{t \blacktriangleright t' \in T : t \vdash^*_{T+R-t\blacktriangleright t'} t'\}).$$

8.4 Utility

Alice uses measures for keeping track of the historic utility of her different beliefs. These measures play a role when she determines which rules to keep and which ones to remove from her belief set. The following measures are used for distinguishing between good and bad rules R:

Historic desired contribution: How many times $k \leq n$ has Alice used R to compute a positive fact $F_k = t \vdash t'$?

Historic undesired contribution: How many times $k \leq n$ has Alice used R to compute a negative fact $F_k = t \nvdash t'$?

8.5 Irredundancy

Finally, Alice keeps her belief set irredundant in the following sense:

Definition 53 (weak irredundancy) *The theory T is* weakly irredundant *if for all rules $R \in T$ of the form $t \rhd t'$ or $t \blacktriangleright t'$ we have $t \nvdash^*_{T-R} t'$.*

We use the term weakly irredundant here to emphasize that the property concerns the relation $\vdash^*_T$ rather than $\vdash_T$.

9. How Alice learns

In this section we will show how Alice updates her belief set T_n. Alice defines T_{n+1} by using one of two mechanisms:

Exogenic update. Define T_{n+1} based on F_n. If $F_n = t \vdash t'$ and $t \nvdash^*_{T_n} t'$, add a rule so that $t \vdash^*_{T_{n+1}} t'$ if prioritized. If $F_n = t \nvdash t'$ and $t \vdash^*_{T_n} t'$, remove one or more rules so that $t \nvdash^*_{T_{n+1}} t'$. Add F_n if prioritized.

Endogenic update. Define T_{n+1} based on T_n. If prioritized, add open rule R with the properties that (i) several (as defined by a parameter) closed instances of R are bounded-computable in T_n and (ii) no small (as defined by a parameter) counterexample to R is bounded-computable in T_n.

	$F_n = t \vdash t'$	$F_n = t \nvdash t'$
$t \vdash^*_{T_n} t'$	Endogenic update	Exogenic update
$t \nvdash^*_{T_n} t'$	Exogenic update	Endogenic update

Table 3: Criteria for selecting update mechanism depending on F_n and $\vdash^*_{T_n}$. If Alice's beliefs match the observed fact about Wonderland, she makes an endogenic update. Otherwise she makes an exogenic update.

If T_n is "full" in the sense that $card(T_n) = Alice_{LTM} = 200$ rules, no new rule can be added to T_{n+1} before some old rule has been removed. This is a necessary consequence of the finite resources of Alice and any other system with a continuously developing knowledge base. When the knowledge base is full, Alice switches focus from quantity to quality of the rules. An important part of Alice's strategy is that she never stops testing her old open rules with the help of new observations that she makes.

The main algorithms of ALICE IN WONDERLAND are presented in Section B. Table 4 gives several examples of how Alice operates in learning mode.

10. How Alice performs

In this section we give some examples of domains that ALICE IN WONDERLAND is able to learn and problems in those domains that the system can solve.

ALICE IN WONDERLAND typically starts from a state of no knowledge and learns from a random stream of facts taken from an arbitrary domain. Learning is assessed in terms of acquiring new knowledge, i.e., facts other than those that are just memorized. This occurs when ALICE IN WONDERLAND can form and verify hypotheses about a domain and make use of them to solve unseen problems. This does not necessarily mean it can acquire a complete belief set about a domain, even in cases when such belief sets exist.

An example of how the system starts from scratch and gradually constructs a set of beliefs is shown in Table 4. Examples of how it solves problems are shown in Tables 1–2.

Here are some examples of domains that ALICE IN WONDERLAND was able to learn from random streams of facts:

Simple mathematics It was able to learn a complete theory of integer addition and multiplication. For instance, the system found the commutative laws, e.g. $x+y = y+x$, the associative laws, e.g. $x*(y*z) = (x*y)*z$, the distributive law $x*(y+z) = (x*y)+(y*z)$ together with the identity axioms, e.g. $x+0 = 0$. It also found complete set of rules for the modulo operators even, odd, and modulo 3, and the ring of polynomials $F_2[X]$. Moreover, it was able to learn patterns in simple number sequences, such as arithmetic and geometric sequences.

Simple logic It can learn axioms of propositional logic from random facts and can prove several of the tautologies listed in Appendix C in Strannegård et al. (2010). Thus it learned several

Fact	Theory update
`2*0` $\vdash$ `0`	Add `2*0` $\blacktriangleright$ `0`.
`3*0` $\vdash$ `0`	Add `3*0` $\blacktriangleright$ `0`.
`2*0` $\vdash$ `0`	Add `x` $\blacktriangleright$ `0`.
`1` $\nvdash$ `0`	Remove `x` $\blacktriangleright$ `0`. Add `1` $\not\triangleright$ `0`.
`2*(3*0)` $\vdash$ `0`	Add `x*0` $\blacktriangleright$ `0`.
`75*0` $\vdash$ `0`	Do nothing.
`0*0` $\vdash$ `0`	Add `0*0` $\blacktriangleright$ `0`.
`1*1` $\vdash$ `1`	Add `1*1` $\blacktriangleright$ `1`.
`1*1` $\vdash$ `1`	Add `x*x` $\blacktriangleright$ `x`.
`2*2` $\nvdash$ `2`	Remove `x*x` $\blacktriangleright$ `x`. Add `2*2` $\not\triangleright$ `2`.
`5+2` $\vdash$ `7`	Add `5+2` $\blacktriangleright$ `7`.
`23` $\nvdash$ `45`	Do nothing.
`f(0)` $\vdash$ `5`	Add `f(0)` $\blacktriangleright$ `5`.
`f(1)` $\vdash$ `7`	Add `f(1)` $\blacktriangleright$ `7`.
`f(1)` $\vdash$ `7`	Add `f(x+1)` $\blacktriangleright$ `f(x)+2`.
`Alice crawls` $\vdash$ `OK`	Add `Alice crawls` $\blacktriangleright$ `OK`.
`Alice runs` $\vdash$ `OK`	Add `runs` $\blacktriangleright$ `crawls`.

Table 4: This table shows an example of a learning process $T_0, \ldots T_{17}$, where $T_0 = \emptyset$. The theories are updated using a mix of endogenic and exogenic updates. Lines 4 and 10 show how Alice deals with inconsistent information. On line 7, Alice saves `0*0` $\blacktriangleright$ `0`, which is redundant since she already believes that `x*0` $\blacktriangleright$ `0`. The reason is that, for all Alice knows, `x*0` $\blacktriangleright$ `0` might have to be abandoned later if a counter-example is found. In that scenario, small table entries like `0*0` $\blacktriangleright$ `0` would become non-redundant again.

tautologies that are useful for proving other tautologies, e.g. $\neg\neg A \leftrightarrow A$. It could also learn quantifier-free first-order logic with function symbols (including constants) and predicate symbols.

Simple grammar In natural languages such as English, it was able to find grammatically equivalent words such as nouns and transitive and intransitive verbs. For instance, it found that horse can be replaced by cow in some simple examples and never saw any example that falsified its assumption that those two words are interchangeable as far as grammatical correctness is concerned. Similarly with run and walk (which can both be transitive as well as intransitive). Using this knowledge, it was able to tentatively judge the grammatical correctness of previously unseen sentences. This is limited to non-ambiguous examples and context-free grammars. The system also manages simple fragments of Swedish and Sindi, in their respective scripts, in a similar manner.

The above results illustrate the versatility of the system. The system does not break any new ground, however, when it comes to learning grammar, arithmetic, or number sequences. The point is that one and the same system is able to start from scratch and learn all those domains and many others. ALICE IN WONDERLAND is equipped with several general mechanisms of learning and

reasoning, as described in the previous sections. It operates with bounded cognitive resources that bring down the computational complexity of the system. This move ensures computability and and distinguishes ALICE IN WONDERLANDfrom systems where bounded resources are not used systematically.

ALICE IN WONDERLAND generalizes several ides that were implemented in our previous systems for inductive learning and automated reasoning, including SEQSOLVER (Strannegård et al., 2013), which outperformed average human score in solving number sequence problems of standard IQ tests, and OCCAM* (Nizamani and Strannegård, 2014), which outscored average human performance in solving tautologies in propositional logic. Those systems were highly specialized and lacked the generality of ALICE IN WONDERLAND. Although we did not formally compare ALICE IN WONDERLAND to those systems on the relevant tasks, our impression is that we did pay a prize for the increased generality in the form of a somewhat poorer performance compared to the specialized systems. Nevertheless, ALICE IN WONDERLAND performs above human level in both logic and arithmetic.

11. Conclusion

We have described the system ALICE IN WONDERLAND that is able to learn autonomously and solve new problems in arbitrary domains. The system is flexible, independent of human interaction, able to start from scratch and learn from arbitrary streams of facts. For evaluation purposes we used a number of randomly generated streams of facts from different domains, but the streams could come from any source, including digital sensors, human entry, or human feedback on behavior.

We used a simple cognitive model that reduces the computational complexity from undecidable to finite. Thus we propose to tackle the combinatorial-explosion problem that arises in, e.g. inductive logic programming, automated theorem proving, and inductive grammar learning. Our theoretical approach was to extend several basic concepts and theorems of logic into a more general setting. Those concepts form the theoretical basis of ALICE IN WONDERLAND, but we hope that they can be fruitfully used in other systems as well.

Our results show that ALICE IN WONDERLAND is able to learn multiple domains from random streams of facts and challenge average human problem solvers in domains such as propositional logic and simple arithmetic. We find the approach to AGI that was explored in this paper to be 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.

References

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

Bach, J. 2012. MicroPsi 2: The Next Generation of the MicroPsi Framework. In *Artificial General Intelligence*. Springer. 11–20.

Carroll, L. 2000. *Alice's Adventures in Wonderland*. Broadview Press.

Clarke, D.; Whitney, H.; Sutton, G.; and Robert, D. 2013. Detection and learning of floral electric fields by bumblebees. *Science* 340(6128):66–69.

Dowe, D. L.; Hernández-Orallo, J.; and Das, P. K. 2011. Compression and intelligence: social environments and communication. In *Artificial General Intelligence*. Springer. 204–211.

Goertzel, B.; Pennachin, C.; and Geisweiller, N. 2014. The OpenCog Framework. In *Engineering General Intelligence, Part 2*. Springer. 3–29.

Gould, J. L.; Gould, C. G.; et al. 1988. *The honey bee*. Scientific American Library.

Hansson, S. O.; Fermé, E. L.; Cantwell, J.; and Falappa, M. A. 2001. Credibility Limited Revision. *The Journal of Symbolic Logic* 66(04):1581–1596.

Katayama, S. 2015. Towards Human-Level Inductive Functional Programming. In *Artificial General Intelligence*. Springer. 111–120.

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

Laird, J. 2012. *The Soar cognitive architecture*. MIT Press.

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

Negri, S.; Von Plato, J.; and Ranta, A. 2008. *Structural proof theory*. Cambridge University Press.

Nizamani, A. R., and Strannegård, C. 2014. Learning Propositional Logic From Scratch. In *The 28th annual workshop of the Swedish Artificial Intelligence Society (SAIS)*.

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

Popper, K. R. 1972. *Objective knowledge: An evolutionary approach*. Clarendon Press Oxford.

Rosenbloom, P. S. 2013. The Sigma Cognitive Architecture and System. *AISB Quarterly* 136:4–13.

Simon, H. A. 1982. *Models of Bounded Rationality: Empirically Grounded Economic Reason*, volume 3. MIT press.

Steunebrink, B. R.; Koutník, J.; Thórisson, K. R.; Nivel, E.; and Schmidhuber, J. 2013. Resource-bounded machines are motivated to be effective, efficient, and curious. In *Artificial General Intelligence*. Springer. 119–129.

Strannegård, C.; Ulfsbäcker, S.; Hedqvist, D.; and Gärling, T. 2010. Reasoning Processes in Propositional Logic. *Journal of Logic, Language and Information* 19(3):283–314.

Strannegård, C.; Nizamani, A. R.; Sjöberg, A.; and Engström, F. 2013. Bounded Kolmogorov Complexity Based on Cognitive Models. In Kühnberger, K. U.; Rudolph, S.; and Wang, P., eds., *Proceedings of AGI 2013, Beijing, China*. Springer.

Turing, A. 1950. Computing machinery and intelligence. *Mind* 59(236):433–460.

Von Glasersfeld, E. 1995. *Radical Constructivism: A Way of Knowing and Learning. Studies in Mathematics Education Series: 6.* ERIC.

Wang, P., and Hammer, P. 2015. Assumptions of Decision-Making Models in AGI. In *Artificial General Intelligence*. Springer. 197–207.

Wiese, K. 2002. *The Crustacean Nervous System*. Springer.

Appendix A. Table of symbols

Symbol	Concept	Definition
$\square$	context	Definition 7
$\vdash$	fact	Definition 8
$\models_D$	consequence in domain D	Definition 14
$\sim_D$	similarity in domain D	Definition 16
$\equiv_D$	equivalence in domain D	Definition 17
$\triangleright$	shallow rule	Definition 24
$\blacktriangleright$	deep rule	Definition 24
$deep(T)$	deep rules of theory T	Definition 27
$cl(T)$	closed rules of theory T	Definition 28
$\vdash_T$	computability in theory T	Definition 30
$\vdash_T^*$	bounded computability in theory T	Definition 37

Table 5: Symbols of the paper in order of appearance.

Appendix B. Algorithms of ALICE IN WONDERLAND

Algorithm 1 shows the learning algorithm of the program. The preference orders of Algorithm 5 are used for determining which rules should be added to or removed from T_n. Update mechanisms are invoked as described in Table 3.

Algorithm 1: Main learning algorithm of ALICE IN WONDERLAND

Input: F_n ($t \vdash t'$ or $t \nvdash t'$)
Data: T_n: agent's theory containing `rules`
Data: `candidates`: candidate axioms
Output: T_{n+1}
begin
  case $F_n = t \nvdash t'$
    if *there exists a bounded computation from t to t′* **then**
      remove from T_n the smallest subset of rules used in the bounded computation, such that $t \vdash t'$ becomes incomputable
    end
    if T_n *contains* $t' \blacktriangleright t$ **then** change it to $t' \rhd t$
    return T_n
  end
  case $F_n = t \vdash t'$
    abstractions := union (abstractions of t) (abstractions of t')
    updateGraph candidates abstractions
    if $t \vdash t'$ *is solved (unless* $t \vdash t'$ *is a small fact)* **then**
      return T_n
    else search for delta:
      `mixed` := axioms containing one constant and one variable, and with at least two instances in T_n (*e.g.*, `x*0=0`)
      `mixedPlus` := axioms containing one constant and >1 variables, and with at least two instances in T_n
      `algebraic` := valid axioms from candidates (*e.g.*, `x+y=y+x`)
      `synonyms` := synonymous words (in language domain)
      `delta′ := t ▶ t′ ++ mixed ++ mixedplus ++ algebraic ++ synonyms`
      `delta` := **chooseSimplest**
                 . map computePerformance
                 . filter (notProvable
                              and not already in T_n
                              and not one of recently removed axioms
                              and no divergence)
                 $ `delta′`
      T_{n+1} := **difference** (**union** `delta` T_n) (redundant axioms from T_n)
      return T_{n+1}
    end
  end
end

Algorithm 2: Algorithm for updating the candidate axioms

Function: `updateGraph`
Input: abstractions : abstractions of t and t', *e.g.*, $x * y$
Data: candidates : candidate axioms, *e.g.*, $x + y \blacktriangleright y + x$
Output: updated candidates
begin
  foreach *a in abstractions* **do**
    foreach *c in existing abstractions in candidates* **do**
      add $a \blacktriangleright c$ to candidates (*e.g.*, $x + y \blacktriangleright x * y$)
    end
  end
  foreach $c \blacktriangleright c'$ *in candidates* **do**
    if $c \blacktriangleright c'$ *is Invalid* **then**
      do nothing
    else
      if c *or* c' *is an abstraction of* t **then**
        σ := substitution of t
        if $\sigma(c)$ *and* $\sigma(c')$ *converge* **then** add σ to the evidence for $c \blacktriangleright c'$
      end
      σ := random substitution
      if $\sigma(c)$ *and* $\sigma(c')$ *converge* **then** add σ to the evidence for $c \blacktriangleright c'$
      if ***satisfy*** *(evidence for* $c \blacktriangleright c'$*)* **then** set $c \blacktriangleright c'$ as Valid
    end
  end
end

Algorithm 3: Performance computing function

Function: `computePerformance`
Input: delta : an axiom
begin
  computation := find computation of $t \vdash t'$ using union of T_n and `delta`
  usefulness := whether computation contains `delta`
  utility := utility of $t \vdash t'$
  compSize := size of bounded computation of $t \vdash t'$
  return *(delta,usefulness,utility,compSize)*
end

Algorithm 4: Algorithm for measuring evidence of a candidate axiom for satisfiability.

Function: `satisfy`
Input: n : number of variable types in candidate axiom $c \vdash c'$
Input: evidence : set of substitutions for which $c \vdash c'$ converges
begin
  case *n == 1*
    requiredEvidence := if concepts > 2 then 3 else 2
    return *(length evidence >= requiredEvidence)*
  case *n > 1*
    evidences' := for each value of first variable (e.g. x), get values of remaining variables
    result := [(satisfy (n-1) ev) is True | ev <- evidences']
    return *(result >= 2)*
end

Algorithm 5: Algorithm for selecting a single axiom from the valid candidate axioms

Function: `chooseSimplest`
Input: list of candidate axioms
Output: most preferred candidate axiom
begin
  prefer useful `// one that is used in the solution of` $t \vdash t'$
    then prefer small fact `// e.g.,` $2+3=5$
      then prefer smallest in size
        then prefer max utility `// utility of` $t \vdash t'$
          then prefer maximum variable tokens
            then prefer minimum variable types
end