Mach Learn (2014) 94:25–49
DOI 10.1007/s10994-013-5358-3

Meta-interpretive learning: application to grammatical inference

Stephen H. Muggleton · Dianhuan Lin · Niels Pahlavi · Alireza Tamaddoni-Nezhad

Received: 15 December 2012 / Accepted: 12 April 2013 / Published online: 1 May 2013

Abstract Despite early interest Predicate Invention has lately been under-explored within ILP. We develop a framework in which predicate invention and recursive generalisations are implemented using abduction with respect to a meta-interpreter. The approach is based on a previously unexplored case of Inverse Entailment for Grammatical Inference of Regular languages. Every abduced grammar H is represented by a conjunction of existentially quantified atomic formulae. Thus $\neg H$ is a universally quantified clause representing a denial. The hypothesis space of solutions for $\neg H$ can be ordered by θ-subsumption. We show that the representation can be mapped to a fragment of Higher-Order Datalog in which atomic formulae in H are projections of first-order definite clause grammar rules and the existentially quantified variables are projections of first-order predicate symbols. This allows predicate invention to be effected by the introduction of first-order variables. Previous work by Inoue and Furukawa used abduction and meta-level reasoning to invent predicates representing propositions. By contrast, the present paper uses abduction with a meta-interpretive framework to invent relations. We describe the implementations of Meta-interpretive Learning (MIL) using two different declarative representations: Prolog and Answer Set Programming (ASP). We compare these implementations against a state-of-the-art ILP system MC-TopLog using the dataset of learning Regular and Context-Free grammars as well learning a simplified natural language grammar and a grammatical description of a staircase. Experiments indicate that on randomly chosen grammars, the two implementations have significantly higher accuracies than MC-TopLog. In terms of running time, Metagol is overall fastest in these tasks. Experiments indicate that the Prolog implementation is competitive with the ASP one due to its ability to encode a strong procedural bias. We demonstrate that MIL can be applied to learning natural grammars. In this case experiments indicate that increasing the available background knowledge, reduces the running time. Additionally $\mathrm{ASP_M}$ (ASP using a meta-interpreter) is shown to have a speed advantage over Metagol when background knowledge is sparse. We also demonstrate that by combining $\mathrm{Metagol}_R$

Editors: Fabrizio Riguzzi and Filip Zelezny.

S.H. Muggleton (✉) · D. Lin · N. Pahlavi · A. Tamaddoni-Nezhad
Department of Computing, Imperial College London, London, UK
e-mail: s.muggleton@imperial.ac.uk

(Metagol with a Regular grammar meta-interpreter) and Metagol$_{CF}$ (Context-Free meta-interpreter) we can formulate a system, Metagol$_{RCF}$, which can change representation by firstly assuming the target to be Regular, and then failing this, switch to assuming it to be Context-Free. Metagol$_{RCF}$ runs up to 100 times faster than Metagol$_{CF}$ on grammars chosen randomly from Regular and non-Regular Context-Free grammars.

Keywords Inductive logic programming · Meta-interpretative learning · Predicate invention · Recursion · Grammatical inference

1 Introduction

Consider the problem of using an ILP system to learn a Regular grammar which accepts all and only those binary sequences containing an even number of 1*s* (see Fig. 1). Since the 1950s automaton-based learning algorithms have existed (Moore 1956) which inductively infer Regular languages, such as *Parity*, from positive and negative examples. If we try to learn *Parity* using an ILP system the obvious representation of the target would be a Definite Clause Grammar (DCG) (see Fig. 1a). However, if the ILP system were provided with examples for the predicate q_0 then the predicate q_1 would need to be invented since the only single state finite acceptor consistent with the examples would accept all finite strings consisting of 0*s* and 1*s*. It is widely accepted that Predicate Invention is a hard and under-explored topic within ILP (Muggleton et al. 2011), and indeed state-of-the-art ILP systems, including MC-TopLog (Muggleton et al. 2012) and Progol (Muggleton 1995; Muggleton and Bryant 2000), are unable to learn grammars such as *Parity* in the form of a DCG using only first-order (non-metalogical) background knowledge since these systems do not support Predicate Invention. However, note that in Fig. 1a each clause of the DCG has one of the following two forms.

$$Q([], []) \leftarrow$$
$$Q([C|x], y) \leftarrow P(x, y)$$

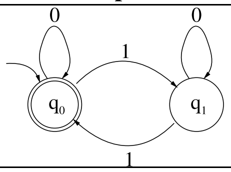
where Q, C, P are the only symbols which vary between the clauses. Figure 1b shows how these two forms of clauses above can be captured within the two clauses of a recursive meta-interpreter *parse/3* which uses the auxiliary predicates *acceptor/1* and *delta1/3*[1][2] to instantiate the predicate symbols and constants from the original DCG. The predicates *acceptor/1* and *delta1/3* can each be interpreted as Higher-Order Datalog (Muggleton and Pahlavi 2012) predicates since they take arguments which are predicate symbols q_0, q_1 from the DCG. By making *acceptor/1* and *delta1/3* abducible, *Parity*, and indeed any other Regular grammar, could in principle be learned from ground instances of *parse/1* using abduction. The paper explores this form of learning with respect to a meta-interpreter.

We show that such abductively inferred grammars are a special case of Inverse Entailment. We also show that the hypothesis space forms a lattice ordered by subsumption. The extensions of this use of abduction with respect to a meta-interpreter lead to a new class of inductive algorithm for learning Regular and Context-Free languages. The new approach

[1]Note that in the theory of automata (Hopcroft and Ullman 1979) *delta1/3* corresponds to the transition function of the finite acceptor shown in Fig. 1a.

[2]Considering *delta1/3* as an arity 3 ground relation, if c, k are bounds on the number of terminals and non-terminals respectively then the number of possible definitions for *delta1/3* is 2^{ck^2}.

(a)

Finite acceptor	Production rules	Definite Clause Grammar (DCG)
	$q_0 \rightarrow$	$q_0([], []) \leftarrow$
	$q_0 \rightarrow 0\, q_0$	$q_0([0\|A], B) \leftarrow q_0(A, B)$
	$q_0 \rightarrow 1\, q_1$	$q_0([1\|A], B) \leftarrow q_1(A, B)$
	$q_1 \rightarrow 0\, q_1$	$q_1([0\|A], B) \leftarrow q_1(A, B)$
	$q_1 \rightarrow 1\, q_0$	$q_1([1\|A], B) \leftarrow q_0(A, B)$

(b)

E^+	E^-	Meta-interpreter	Ground facts
λ	1	$parse(S) \leftarrow parse(q0, S, [])$.	
0	01		$acceptor(q0) \leftarrow$
00	10	$parse(Q, [], []) \leftarrow acceptor(Q)$.	$delta1(q0, 0, q0) \leftarrow$
11	001	$parse(Q, [C\|X], Y) \leftarrow$	$delta1(q0, 1, q1) \leftarrow$
000	010	$delta1(Q, C, P)$,	$delta1(q1, 0, q1) \leftarrow$
011	100	$parse(P, X, Y)$.	$delta1(q1, 1, q0) \leftarrow$
101	111		

Fig. 1 (**a**) Parity acceptor with associated production rules, DCG; (**b**) positive examples (E^+) and negative examples (E^-), Meta-interpreter and ground facts representing the Parity grammar

blurs the normal distinctions between abductive and inductive techniques (see Flach and Kakas 2000). Usually abduction is thought of as providing an explanation in the form of a set of ground facts while induction provides an explanation in the form of a set of universally quantified rules. However, the meta-interpreter in Fig. 1b can be viewed as projecting the universally quantified rules in Fig. 1a onto the ground facts associated with *acceptor/1* and *delta1/3* in Fig. 1b. In this way abducing these ground facts with respect to a meta-interpreter is equivalent to induction, since it is trivial to map the ground *acceptor/1* and *delta1/3* facts back to the original universally quantified DCG rules.

In this paper, we show that the MIL framework can be directly implemented using declarative techniques such as Prolog and Answer Set Programming (ASP). In this way, the search for an hypothesis in a learning task is delegated to the search engine in Prolog or ASP. Although existing abductive systems can achieve predicate invention if loaded with the meta-interpreter introduced in this paper, a direct implementation of MIL has the following advantages.

1. As a declarative machine learning (De Raedt 2012) approach, it can make use of the advances in solvers. For example, techniques ASP solvers such as Clasp (Gebser et al. 2007) compete favourably in international competitions. Recently Clasp has been extended to Unclasp (Andres et al. 2012) which is highly efficiency for optimisation tasks. This advance is exploited in the experiments of this paper, as we use Unclasp for our experiments.
2. As demonstrated by the experiments in this paper, direct implementation of the approach using a meta-interpreter has increased efficiency due to an ordered search in the case of Prolog and effective pruning in the case of ASP. While existing abductive systems like SOLAR (Nabeshima et al. 2010), A-System (Kakas et al. 2001) and MC-TopLog do not have an ordered search, but instead enumerate all hypotheses that are consistent with the data.
3. The resulting hypotheses achieve higher predictive due to global optimisation, as opposed to the greedy covering algorithm used in many systems including MC-TopLog.

The paper is structured as follows. Section 2 introduces the theoretical framework for MIL and its application to grammatical inference. We then describe implementations for a variant of Metagol, ASP$_M$ (ASP using a meta-interpreter). In Sect. 4 the performance of these systems is compared experimentally against MC-Toplog on Regular and Context-Free grammar learning problems. In Sect. 5 we describe related work. Lastly we conclude and describe directions for further work in Sect. 6.

2 MIL framework

2.1 Logical notation

A variable is represented by an upper case letter followed by a string of lower case letters and digits. A function symbol or predicate symbol is a lower case letter followed by a string of lower case letters and digits. The arity of a function or predicate symbol is the number of arguments it takes. A constant is a function or predicate symbol which has arity zero. Variables and constants are terms, and a function symbol immediately followed by a bracketed n-tuple of terms is a term. Thus $f(g(X), h)$ is a term when f, g and h are function symbols and X is a variable. A predicate symbol immediately followed by a bracketed n-tuple of terms is called an atomic formula. The negation symbol is $\neg$. Both A and $\neg A$ are literals whenever A is an atomic formula. In this case A is called a positive literal and $\neg A$ is called a negative literal. A finite set (possibly empty) of literals is called a clause. A clause represents the disjunction of its literals. Thus the clause $\{A_1, A_2, \ldots \neg A_i, \neg A_{i+1}, \ldots\}$ can be equivalently represented as $(A_1 \vee A_2 \vee \ldots \neg A_i \vee \neg A_{i+1} \vee \ldots)$ or $A_1, A_2, \ldots \leftarrow A_i, A_{i+1}, \ldots$. A Horn clause is a clause which contains at most one positive literal. A Horn clause is unit if and only if it contains exactly one literal. A denial or goal is a Horn clause which contains no positive literals. A definite clause is a Horn clause which contains exactly one positive literal. The positive literal in a definite clause is called the head of the clause while the negative literals are collectively called the body of the clause. A unit clause is positive if it contains a head and no body. A unit clause is negative if it contains one literal in the body. A set of clauses is called a clausal theory. A clausal theory represents the conjunction of its clauses. Thus the clausal theory $\{C_1, C_2, \ldots\}$ can be equivalently represented as $(C_1 \wedge C_2 \wedge \cdots)$. A clausal theory in which all predicates have arity at most one is called monadic. A clausal theory in which all predicates have arity at most two is called dyadic. A clausal theory in which each clause is Horn is called a Horn logic program. A logic program is said to be definite in the case it contains only definite clauses. A logic program is said to be a Datalog program if it contains no function symbols other than constants. A Datalog program is said to be higher-order in the case that it contains at least one constant predicate symbol which is the argument of a term. Literals, clauses and clausal theories are all well-formed-formulae (wffs) in which the variables are assumed to be universally quantified. Let E be a wff or term. E is said to be ground if and only if it contains no variables. The process of replacing (existential) variables by constants is called Skolemisation. The unique constants are called Skolem constants. Let C and D be clauses. We say that $C \succeq_\theta D$ or C θ-subsumes D if and only if there exists a substitution θ such that $C\theta \subseteq D$.

2.2 Formal language notation

Let Σ be a finite alphabet. Σ^* is the infinite set of strings made up of zero or more letters from Σ. λ is the empty string. uv is the concatenation of strings u and v. $|u|$ is the length of string u. A language L is any subset of Σ^*. Let ν be a set of non-terminal symbols disjoint from Σ. A production rule $r = LHS \rightarrow RHS$ is well-formed in the case that $LHS \in (\nu \cup \Sigma)^*$, $RHS \in (\nu \cup \Sigma \cup \lambda)^*$ and when applied replaces LHS by RHS in a given string. A grammar G is a pair $\langle s, R\rangle$ consisting of a start symbol $s \in \nu$ and a finite set of production rules R. A grammar is Regular Chomsky-normal in the case that it contains only production rules of the form $S \rightarrow \lambda$ or $S \rightarrow aB$ where $S, B \in \nu$ and $a \in \Sigma$. A grammar is Linear Context-Free in the case that it contains only Regular Chomsky-normal production rules or rules of the form $S \rightarrow Ab$ where $S, A \in \nu$ and $b \in \Sigma$. A grammar is Context-Free in the case that it contains only Linear Context-Free Chomsky-normal production rules or rules of the form $S \rightarrow AB$ where $S, A, B \in \nu$.[3] A Context-Free grammar is said to be deterministic in the case that it does not contain two Regular Chomsky-normal production rules $S \rightarrow aB$ and $S \rightarrow aC$ where $B \neq C$. A sentence $\sigma \in \Sigma^*$ is in $L(G)$ iff given a start symbol $S \in \nu$ there exists a sequence of production rule applications $S \rightarrow_{R_1} \cdots \rightarrow_{R_n} \sigma$ where $R_i \in G$. A language L is Regular, Linear Context-free or Context-Free in the case there exists a grammar G for which $L = L(G)$ where G is Regular, Linear Context-Free or Context-Free respectively. According to the Context-Free Pumping Lemma (Hopcroft and Ullman 1979), if a language L is Context-Free, then there exists some integer $p \geq 1$ such that any string s in L with $|s| \geq p$ (where p is a constant) can be written as $s = uvxyz$ with substrings u, v, x, y and z, such that $|vxy| \leq p$, $|vy| \geq 1$ and uv^nxy^nz is in L for every integer $n \geq 0$.

2.3 Framework

The Meta-Interpretive Learning (MIL) setting is a variant of the normal setting for ILP.

Definition 1 (Meta-Interpretive Learning setting) A Meta-Interpretive Learning (MIL) problem consists of *Input* $= \langle B, E\rangle$ and *Output* $= H$ where the background knowledge $B = B_M \cup B_A$. B_M is a definite logic program[4] representing a meta-interpreter and B_A and H are ground definite Higher-Order Datalog programs consisting of positive unit clauses. The predicate symbol constants in B_A and H are represented by Skolem constants. The examples are $E = \langle E^+, E^-\rangle$ where E^+ is a ground logic program consisting of positive unit clauses and E^- is a ground logic program consisting of negative unit clauses. The *Input* and *Output* are such that $B, H \models E^+$ and for all e^- in E^-, $B, H \not\models e^-$.

Inverse Entailment can be applied to allow H to be derived from B and E^+ as follows.

$$\begin{gathered} B, H \models E^+ \\ B, \neg E^+ \models \neg H \end{gathered} \tag{1}$$

Since both H and E^+ can each be treated as conjunctions of ground atoms containing Skolem constants in place of existential variables, it follows that $\neg H$ and $\neg E^+$ are universally quantified denials where the variables come from replacing Skolem constants by unique variables. We now define the concept of a Meta-interpretive learner.

[3]This is an adaptation of Chomsky-normal form Context-free, which only permits productions of the form $S \rightarrow \lambda$, $S \rightarrow a$ and $S \rightarrow AB$.

[4]Note that the meta-interpreter shown in Fig. 1b is a definite logic program. Such a meta-interpreter is only a part of implementations such as those described later in Sect. 3. Within such an implementation negation-by-failure is used to implement operations such as abduction, so the implementation as a whole is not a definite logic program. However, this does not affect this definition or the later propositions which use it.

E^+	$\neg E^+$	E^-
$parse([]) \leftarrow$	$\leftarrow \quad parse([]),$	$\leftarrow parse([1])$
$parse([1,1]) \leftarrow$	$parse([1,1]),$	$\leftarrow parse([0,1])$
$parse([0,1,1]) \leftarrow$	$parse([0,1,1]),$	$\leftarrow parse([1,0])$
$parse([1,0,1]) \leftarrow$	$parse([1,0,1]),$	$\leftarrow parse([0,0,1])$
$parse([1,1,0]) \leftarrow$	$parse([1,1,0]).$	$\leftarrow parse([1,1,1])$

H	$\neg H$
$acceptor(\$0) \leftarrow$	$\leftarrow \quad acceptor(Q0),$
$delta1(\$0,0,\$0) \leftarrow$	$delta1(Q0,0,Q0),$
$delta1(\$0,1,\$1) \leftarrow$	$delta1(Q0,1,Q1),$
$delta1(\$1,0,\$1) \leftarrow$	$delta1(Q1,0,Q1),$
$delta1(\$1,1,\$0) \leftarrow$	$delta1(Q1,1,Q0).$

Fig. 2 Parity example where B_M is the Meta-interpreter shown in Fig. 1b, $B_A = \emptyset$ and E^+, $\neg E^+$, E^-, H, $\neg H$, are as shown above. '\$0' and '\$1' in H are Skolem constants replacing existentially quantified variables

Definition 2 (Meta-interpretive learner) Let $\mathcal{H}_{B,E}$ represent the complete set of abductive hypotheses H for the MIL setting of Definition 1. Algorithm A is said to be a Meta-interpretive learner iff for all B, E such that H is the output of Algorithm A given B and E as inputs, it is the case that $H \in \mathcal{H}_{B,E}$.

Example 1 (Parity example) Let $B = \langle B_M, B_A \rangle$, $E = \langle E^+, E^- \rangle$ and $H \in \mathcal{H}_{B,E}$ represents the parity grammar. Figure 2 shows H as a possible output of a Meta-interpretive learner.

Note that this example of abduction produces Predicate Invention by introducing Skolem constants representing new predicate symbols. By contrast an ILP system such as Progol uses Inverse Entailment (Muggleton 1995) to construct a single clause from a single example, while a Meta-interpretive learner uses Inverse Entailment to construct the set of all clauses H as the abductive solution to a single goal $\neg E^+$ using E^- as integrity constraints. In the example the hypothesised grammar H corresponds to the first-order DCG from Fig. 1a, which contains both invented predicates and mutual recursion. Neither predicate invention nor mutual recursion can be achieved with DCGs in this way using ILP systems such as Progol or MC-TopLog.

2.4 Lattice properties of hypothesis space

In this section we investigate orderings over MIL hypotheses.

Definition 3 ($\succeq_{B,E}$ relation in MIL) Within the MIL setting we say that $H \succeq_{B,E} H'$ in the case that $H, H' \in \mathcal{H}_{B,E}$ and $\neg H' \succeq_\theta \neg H$.

We now show that $\succeq_{B,E}$ forms a quasi-ordering and a lattice.

Proposition 1 (Quasi-ordering) *Within the MIL setting $\langle \mathcal{H}_{B,E}, \succeq_{B,E} \rangle$ forms a quasi-ordering.*

Proof Follows from the fact that $\langle \{\neg H : H \in \mathcal{H}_{B,E}\}, \succeq_\theta \rangle$ forms a quasi-ordering since each $\neg H$ is a clause (Nienhuys-Cheng and de Wolf 1997). □

Proposition 2 (Lattice) *Within the MIL setting $\langle \mathcal{H}_{B,E}, \succeq_{B,E}\rangle$ forms a lattice.*

Proof Follows from the fact that $\langle\{\neg H : H \in \mathcal{H}_{B,E}\}, \succeq_\theta\rangle$ forms a lattice since each $\neg H$ is a clause (Nienhuys-Cheng and de Wolf 1997). □

We now show that this ordering has a unique top element.

Proposition 3 (Unique $\top$ element) *Within the MIL setting there exists $\top \in \mathcal{H}_{B,E}$ such that for all $H \in \mathcal{H}_{B,E}$ we have $\top \succeq_{B,E} H$ and $\top$ is unique up to renaming of Skolem constants.*

Proof Let $\neg H' = \bigvee_{H\in\mathcal{H}_{B,E}} \neg H$ and $\neg\top = \neg H'\theta_v$ where v is a variable and $\theta_v = \{u/v : u \text{ variable in } \neg H'\}$. By construction for each $H \in \mathcal{H}_{B,E}$ it follows that $\neg\top \succeq_\theta \neg H$ with substitution θ_v. Therefore for all $H \in \mathcal{H}_{B,E}$ we have $\top \succeq_{B,E} H$ and $\top$ is unique up to renaming of Skolem constants. □

This proposition can be illustrated with a grammar example.

Example 2 (Subsumption example) In terms of the Meta-interpreter of Fig. 1a the universal grammar $\{0, 1\}^*$ can be expressed using $\top = \{(acceptor(\$0) \leftarrow), (delta1(\$0, 0, \$0) \leftarrow), (delta1(\$0, 1, \$0) \leftarrow)\}$. Letting H represent the Parity grammar from Example 1 it is clear that $\neg H \succeq_\theta \neg\top$ and so $\top \succeq_{B,E} H$. So unlike the subsumption relation between universally quantified clauses, binding all the (existentially quantified) variables in H to each other produces a maximally general grammar $\top$.

We now show the circumstances under which a unique bottom element of the lattice can be constructed using Plotkin's lgg algorithm.

Proposition 4 (Unique $\bot$ element) *In the case that $\mathcal{H}_{B,E}$ is finite up to renaming of Skolem constants there exists $\bot \in \mathcal{H}_{B,E}$ such that for all $H \in \mathcal{H}_{B,E}$ we have $H \succeq_{B,E} \bot$ and $\bot$ is unique up to renaming of Skolem constants.*

Proof Since $\mathcal{H}_{B,E}$ is finite $\neg\bot = lgg(\{\neg H : H \in \mathcal{H}_{B,E}\})$ where lgg is Plotkin's algorithm for computing the least general generalisation of a set of clauses under subsumption (Plotkin 1969). □

For most purposes the construction of the unique bottom clause is intractable since the cardinality of the lgg clause increases exponentially in the cardinality of $\mathcal{H}_{B,E}$. We now show a method for reducing hypotheses.

2.5 Reduction of hypotheses

Proposition 5 (Logical reduction of hypotheses) *Suppose H' is an hypothesis in the MIL setting and $\neg H$ is the result of applying Plotkin's clause reduction algorithm*[5] (Plotkin 1969) *to $\neg H'$. Then H is a reduced hypothesis equivalent to H'.*

[5]This algorithm iteratively remove logically redundant literals from a clause until no redundant clauses remain.

Language type	**Meta-interpreter**	**Example Grammar**	**L**
(a) R	$parse(S) \leftarrow parse(Q, S, [])$. $parse(Q, X, X) \leftarrow acceptor(Q)$. $parse(Q, [C\|X], Y) \leftarrow delta1(Q, C, P)$, $parse(P, X, Y)$.	$S \rightarrow 0\ S$ $S \rightarrow 1\ T$ $T \rightarrow \lambda$ $T \rightarrow 1\ T$	$0^{+}1^{+}$
(b) CF	$parse(S) \leftarrow parse(Q, S, [])$. $parse(Q, X, X) \leftarrow acceptor(Q)$. $parse(Q, [C\|X], Y) \leftarrow delta1(Q, C, P)$, $parse(P, X, Y)$. $parse(Q, X, Y) \leftarrow delta2(Q, P, C)$, $parse(P, X, [C\|Y])$. $parse(Q, X, Y) \leftarrow delta3(Q, P, R)$, $parse(P, X, Z)$, $parse(R, Z, Y)$.	$S \rightarrow \lambda$ $S \rightarrow T\ S$ $T \rightarrow 0\ U$ $U \rightarrow T\ 1$	$(0^{n}1^{n})^{*}$

Fig. 3 Meta-interpreters, Chomsky-normal form grammars and languages for (**a**) Regular (R) and (**b**) Context-Free (CF) languages

Proof Follows from the fact that $\neg H'$ is θ-subsumption equivalent to $\neg H$ by construction. □

Example 3 (Reduction example) Let $H' = H \cup \{r\}$ where H is the Parity grammar from Fig. 2 and $r = (delta1(\$0, 0, \$2) \leftarrow)$ represents an additional redundant grammar rule. Now Plotkin's reduction algorithm would reduce $\neg H'$ to the equivalent clause $\neg H$ and consequently grammar H is a reduced equivalent form of H.

In the following section we show the existence of a compact bottom hypothesis in the case of MIL for Regular languages.

2.6 Framework applied to grammar learning

Figure 3 shows how the Meta-interpreter for Regular Grammars can be extended to Context-Free Grammars.

The Chomsky language types form an inclusion hierarchy in which Regular $\subseteq$ Context-Free. Algorithms for learning the Regular languages have been widely studied since the 1970s within the topic of Grammatical Inference (de la Higuera 2005). Many of these start with a prefix tree acceptor, and then progressively merge the states.

Proposition 6 (Unique $\perp$ for Regular languages) *Prefix trees act as a compact bottom theory in the MIL setting for Regular languages.*

Proof Follows from the fact that all deterministic Regular grammars which include the positive examples can be formed by merging the arcs of a prefix tree acceptor (Muggleton 1990). Merging the arcs of the prefix tree is achieved by unifying the delta1 atoms in $\neg H$ within the MIL setting. □

Example 4 (Prefix tree) Assume the MIL setting with B_M being the meta-interpreter for Regular languages. Let $E^{+} = \{parse([1, 1]), parse([1, 1, 0])\}$ then $\perp = \{delta1(\$0, 1, \$1), delta1(\$1, 1, \$2), acceptor(\$2), delta1(\$2, 0, \$3), acceptor(\$3)\}$ represents the prefix tree automaton.

```
parse(S,G1,G2) :- parse(s(0),S,[],G1,G2).

parse(Q,X,X,G1,G2) :- abduce(acceptor(Q),G1,G2).
parse(Q,[C|X],Y,G1,G2) :- Skolem(P), abduce(delta1(Q,C,P),G1,G3), parse(P,X,Y,G3,G2).

abduce(X,G,G) :- member(X,G).
abduce(X,G,[X|G]) :- not(member(X,G)).

Skolem(s(0)). Skolem(s(1)). ...
```

Fig. 4 noMetagol$_R$

Proposition 7 ($\perp$ for Context-Free languages) *Any bottom theory $\perp$ for a Context-Free language contains a set of delta*1 *atoms representing a Regular prefix tree.*

Proof Follows from the fact that the Regular subset of MIL hypotheses are all subsumed by $\neg\perp_R$ where $\perp_R$ represents the Regular prefix tree. □

3 Implementations

In this section, we describe the implementations of Meta-interpretive Learning (MIL) using two different declarative languages: Prolog and Answer Set Programming (ASP). The resulting systems are called Metagol[6] and ASP$_{\mathrm{M}}$,[7] respectively.

3.1 Implementation in Prolog

The systems Metagol$_R$, Metagol$_{CF}$, and Metagol$_{RCF}$ are three simple Prolog implementations of MIL.

3.1.1 Metagol$_R$

Before introducing Metagol$_R$, we first explain its simplified version noMetagol$_R$ (non-optimising Metagol$_R$) as shown in Fig. 4. The system noMetagol$_R$ is based on the following abductive variant of the Regular Meta-interpreter from Fig. 3 (the standard definition of member/2 is omitted for brevity).

The abduced atoms are simply accumulated in the extra variables $G1$, $G2$, $G3$. The term $s(0)$ represents the start symbol and a finite set of Skolem constants is provided by the monadic predicate *Skolem*. Hypotheses are now the answer substitutions of a goal such as the following.

```
:-   parse([],[],G1), parse([0],G1,G2), parse([0,0],G2,G3), parse([1,1],G3,G4),    % Pos
     parse([0,0,0],G4,G5), parse([0,1,1],G5,G6), parse([1,0,1],G6,G),
     not(parse([1],G,G)), not(parse([0,1],G,G)).                                    % Neg
```

[6]*Metagol* is MIL encoded within YAP Prolog. The name comes from the combination of *Meta-* and *gol*, where *Meta-* corresponds to the Meta-Interpreter, and *gol* is the reverse of *log* which is short for logic.

[7]The name ASP$_{\mathrm{M}}$ is MIL encoded within an ASP solver.

```
parse(S,G1,G2,S1,S2,K1,K2) :- parse(s(0),S,[],G1,G2,S1,S2,K1,K2).

parse(Q,X,X,G1,G2,S,S,K1,K2) :- abduce(acceptor(Q),G1,G2,K1,K2).
parse(Q,[C|X],Y,G1,G2,S1,S2,K1,K2) :- Skolem(P,S1,S3),
        abduce(delta1(Q,C,P),G1,G3,K3,K2), parse(P,X,Y,G3,G2,S3,S2,K3,K2).

abduce(X,G,G,K,K) :- member(X,G).
abduce(X,G,[X|G],s(K),K) :- not(member(X,G)).

Skolem(s(N),[s(Pre)|SkolemConsts],[s(N),s(Pre)|SkolemConsts]):- N is Pre+1.
Skolem(S,SkolemConsts,SkolemConsts):-member(S,SkolemConsts).
```

Fig. 5 Metagol$_R$

Note that each of the positive examples are provided sequentially within the goal and the resulting grammar is then tested for non-coverage of each of the negative examples. The final grammar returned in the variable G is a solution which covers all positives and none of the negatives. In the case shown above the first hypothesis found by Prolog is as follows.

```
G = [delta1(s(1),0,s(1)),delta1(s(1),1,s(0)),delta1(s(0),1,s(1)),
     delta1(s(0),0,s(0)),acceptor(s(0))]
```

This hypothesis correctly represents the Parity acceptor of Fig. 1. All other consistent hypotheses can be generated by making Prolog backtrack through the SLD proof space.

Metagol$_R$ We will now explain the following procedural biases, which extends noMetagol$_R$ to Metagol$_R$.

Minimal hypothesis Occam's razor suggests to select the shortest hypothesis that fits the data. Therefore we introduce the clause bound into Metagol$_R$ so that the search starts from shorter hypotheses. In Metagol$_R$ (Fig. 5) the variables K, $K1$, $K2$ and $K3$ are related to the clause bound. They are instantiated with Peano numbers ($s(0), s(1), \ldots$) representing a bound on the maximum number of abduced clauses. Thus the second clause of *abduce*/5 fails once $K1$ has a value of 0. $K1$ is iteratively increased until an hypothesis is found within that bound. The search thus guarantees finding an hypothesis with minimal description length.

Specific-to-General Within the MIL setting an hypothesis H_s is said to be more specific than H_g in the case that $\neg H_s \succeq_\theta \neg H_g$, as explained in Sect. 2.4. Therefore H_s is a refinement of H_g by renaming with new Skolem constants. In Metagol$_R$ the Skolem constants are enumerated by the program of *Skolem*/3. The first clause of *Skolem*/3 introduces a new Skolem constant, while the second clause of *Skolem*/3 provides a Skolem constant that has already been used in the deriving hypothesis. Due to Prolog's procedural semantics, the first clause of *Skolem*/3 will be tried before the second one, thus H_s, that is, the one with more Skolem constants, will be considered before H_g.[8] Switching the order of the two clauses in *Skolem*/3 will result in a general-to-specific search. In that case, the universal grammar will be considered first, since it is maximally general and can be expressed with only one Skolem constant (see Example 2).

[8]Provided H_g and H_s are within the same clause bound.

Fig. 6 Metagol$_{RCF}$

```
metagolRCF(G):- metagolR(G).
metagolRCF(G):- metagolCF(G).
```

Metagol$_{CF}$ The Metagol$_{CF}$ system is based on an abductive variant of the Context-Free Meta-interpreter from Fig. 3, though we omit the full Prolog description due to space restrictions. Once more, abduction is carried out with respect to a single goal as in Metagol$_{R}$.

Metagol$_{RCF}$ The Metagol$_{RCF}$ system simply combines Metagol$_{R}$ and Metagol$_{CF}$ sequentially, as shown below in Fig. 6. Due to Prolog's procedural semantics, the hypothesis returned will be Regular in the case Metagol$_{R}$ finds a consistent grammar and otherwise will be the result of Metagol$_{CF}$.

3.2 Implementation in Answer Set Programming (ASP)

Compared to Prolog, ASP not only has advantages in handling non-monotonic reasoning, but also has higher efficiency in tackling search problems (Gebser et al. 2012). The systems ASP$_{\mathrm{M}R}$ and ASP$_{\mathrm{M}CF}$ are two simple ASP implementations of Meta-interpretive learning. Each sequence is encoded as a set of facts. For example, the positive example $posEx(Seq2, [1, 1])$ is encoded in the second line in Fig. 7, where seq2 is the ID of the sequence and the predicate $seqT(SeqID, P, T)$ means the sequence has a terminal T at position P. The meta-interpretive parser uses position to mark a substring, rather than storing the substring in a list. The goal of finding an hypothesis that covers all positive examples and none of the negatives is encoded as an integrity constraint.

ASP$_{\mathrm{M}R}$ The program in Fig. 8 is an ASP implementation of the Regular Meta-interpreter in Fig. 3. It is sectioned into parts describing generating, defining, testing, optimising, and displaying. The generating part specifies the hypothesis space as a set of facts about $delta1/3$ and $acceptor/1$. ASP choice rules are used to indicate that any subset of this set is allowed in the answer sets of this program. The defining part corresponds to the Regular Meta-interpreter. The testing part contains an integrity constraint saying that an answer set of this program should contain production rules which parse all positive examples and no negative examples. The display part restricts the output to containing only predicates $delta1/3$ and $acceptor/1$, which corresponds to the hypothesis.

In order to find a minimal hypothesis like that in Metagol$_{R}$, the optimisation component in ASP is used. Although the use of optimisation increases the computational complexity (Gebser et al. 2012), it improves[9] the predictive accuracy of the hypothesis. An optimisation statement like the one in Fig. 8 specifies the objective function to be optimised. The weight following each atom is part of the objective function. In our case, the objective function

List	Facts
posEx(e1,[0,0]).	posEx(e1). length(e1,2). seqT(e1,0,0). seqT(e1,1,0).
posEx(e2,[1,0,1]).	posEx(e2). length(e2,3). seqT(e2,0,1). seqT(e2,1,0). seqT(e2,2,1).
negEx(e3,[1]).	negEx(e3). length(e3,1). seqT(e3,0,1).
negEx(e4,[0,1]).	negEx(e4). length(e4,2). seqT(e4,0,0). seqT(e4,1,1).

Fig. 7 ASP representation of examples

[9]Occam's razor suggests that simpler hypotheses have higher predictive power. This is further supported by Sect. 4 about experiments, where the non-minimal hypotheses suggested by MC-TopLog have lower predictive accuracies than the minimal ones hypothesised by ASP$_{\mathrm{M}}$ and Metagol.

```
% Instances
#const maxNumSkolemConstants=1.
Skolem(0..maxNumSkolemConstants).
terminal(0;1).

% Generate: specify the hypothesis space
{acceptor(NT):Skolem(NT)}.
{delta1(NT1,T,NT2):Skolem(NT1):terminal(T):Skolem(NT2)}.

% Defining Part
parse(ExID,MaxLengh,MaxLengh,NT):- length(ExID,MaxLengh),acceptor(NT).
parse(ExID,Position1,Position2,NT1):- seqT(ExID,Position1,T),
delta1(NT1,T,NT2), parse(ExID,Position1+1,Position2,NT2).

% Integrity constraint
:- negEx(ExID),length(ExID,MaxLengh),parse(ExID,0,MaxLengh,0)).
:- posEx(ExID),length(ExID,MaxLengh),not parse(ExID,0,MaxLengh,0).

% Optimisation
#minimize [delta1(NT1,T,NT2):Skolem(NT1):terminal(T):Skolem(NT2)=1,
acceptor(NT):Skolem(NT)= 1].

% Displaying
#hide.
#show delta1/3.
#show acceptor/1.
```

Fig. 8 ASP$_{\mathrm{M}R}$

corresponds to the description length of an hypothesis. Therefore the weight is set to 1 for each atom, meaning the description length of a unit clause is 1.

Most ASP solvers do not support variables directly, therefore a grounder is needed for transforming the input program with first-order variables into an equivalent ground program. Then an ASP solver can be applied to find an answer set that satisfies all the constraints. The hypothesised grammar will be part of the returned answer set. In the case shown above the first hypothesis returned by ASP is the same as the one found by Metagol and correctly represents the Parity acceptor of Fig. 1.

ASP solvers use efficient constraint handling techniques to efficiently find stable models known as answer sets. This computational mechanism is very different from that of Prolog, leading to their different implementations, in particular, in the use of iterative deepening. In addition, the bound on clauses puts an implicit limit on Skolem constants, since the number of Skolem constants in a derived hypothesis is at most the number of clauses it contains. Therefore Metagol$_R$ is immune to the number of Skolem constants pre-specified in the background knowledge. By contrast, ASP$_{\mathrm{M}R}$ is largely affected by the number of Skolem constants due to its bottom-up search. Therefore ASP$_{\mathrm{M}R}$ has to put an explicit bound on the number of Skolem constants. More specifically, the second line of 'Generate' $\{delta1(NT1, T, NT2) : Skolem(NT1) : terminal(T) : Skolem(NT2)\}$ has a default size of $T * NT^2$, where T corresponds to the number of terminals and NT denote the number of Skolem constants. While a cardinality constraint on this set does not always reduce the search space, because it can lead to a quadratic blow-up in search space (Gebser et al. 2012) when the cardinality constraint is translated into normal logic program during the grounding stage. Additionally, ASP solvers' build-in optimisation component is handy for finding a global minimal hypothesis. Thus $\mathrm{ASP}_{\mathrm{M}R}$ does not use iterative deepening on the clause bound like that in $\mathrm{Metagol}_R$ for finding a global minimal hypothesis.

$ASP_{\mathrm{M}CF}$ Similar to $\mathrm{Metagol}_{CF}$, the $\mathrm{ASP}_{\mathrm{M}CF}$ system is based on a variant of the Context-Free Meta-interpreter from Fig. 3. However, there is no equivalent ASP implementation to $\mathrm{Metagol}_{RCF}$. Since $\mathrm{Metagol}_{RCF}$ exploits the procedural semantics of Prolog programs, while there is no similar procedural semantics for ASP programs.

4 Experiments

In this section we describe experiments on learning Regular, Context-Free and a simplified natural language grammar. It was shown in Sect. 1 that ILP systems cannot learn grammars in a DCG representation with predicate invention. However, an ILP system given a meta-interpreter as part of background knowledge becomes capable of doing predicate invention. In the experiments described below, the performance of a state-of-the-art ILP system MC-TopLog, loaded with suitable meta-interpretive background, is compared against variants of Metagol and $\mathrm{ASP}_{\mathrm{M}}$ as described in Sect. 3. MC-TopLog is chosen for this comparison since it can learn multiple dependent clauses from examples (unlike say Progol). This is a necessary ability for grammar learning tasks. In the final experiment we show how MIL can be used to learn a definition of a staircase. This indicates the applicability of MIL in more general learning applications beyond grammar learning. All datasets and learning systems used in these experiments are available at http://ilp.doc.ic.ac.uk/metagol.

4.1 Learning regular languages

We investigate the following Null hypotheses.

Null Hypothesis 1.1 $\mathrm{Metagol}_R$, $\mathrm{ASP}_{\mathrm{M}R}$ and a state-of-the-art ILP system cannot learn randomly chosen Regular languages.

Null Hypothesis 1.2 $\mathrm{Metagol}_R$ and $\mathrm{ASP}_{\mathrm{M}R}$ cannot outperform a state-of-the-art ILP system on learning randomly chosen Regular languages.

Null Hypothesis 1.3 $\mathrm{Metagol}_R$ can not outperform $\mathrm{ASP}_{\mathrm{M}R}$ on learning randomly chosen Regular languages.

4.1.1 Materials and methods

Randomly chosen deterministic Regular grammars were generated by sampling from a Stochastic Logic Program (SLP) (Muggleton 1996) which defined the space of target grammars. More specifically, the SLP used for sampling consists of a meta-interpreter and all possible grammars. Then the following steps were conducted. Firstly, an integer i $(1 \leq i \leq 3)$ was randomly sampled. This integer corresponds to the number of seed examples.[10] Secondly, the query "sample(parse(Seq,Grammar))' returned one sequence as well as the grammar that parse this sequence. Thirdly, the grammars were aggregated by issuing the query

[10]The parsing of seed examples requires all rules in the grammar.

Table 1 Average and Maximum lengths of sampled examples for datasets R1, R2, CFG3 and CFG4

	RG1	RG2	CFG3	CFG4
Average ± STD	6.15 ± 4.06	11.43 ± 10.47	5.89 ± 3.08	11.02 ± 9.79
Maximum	15	78	15	68

'sample(parse(Seq,Grammar))" i times. Finally, each generated grammar was reduced using Plotkin's reduction algorithm (see Sect. 2.5) to remove redundancy and equivalent non-terminals. Non-deterministic and finite language grammars were discarded. Sampling of examples was also done using an SLP. Sampling was with replacement.

In this experiment, we used two different datasets sampled from different distributions. In dataset RG1, the examples were randomly chosen from Σ^* for $\Sigma = \{a, b\}$, while in RG2 $\Sigma = \{a, b, c\}$. RG2 has longer sequence lengths, as shown by Table 1. Both datasets contains 200 randomly chosen Regular grammars. We compared the performance of Metagol$_R$, ASP$_{\mathrm{M}R}$ and MC-TopLog on learning Regular grammars using RG1. Only Metagol$_R$ and ASP$_{\mathrm{M}R}$ were compared on RG2, since MC-TopLog failed to terminate due to the longer sequence examples. The performance was evaluated on predictive accuracies and running time.[11] The results were averaged over 200 randomly sampled grammars. For each sample, we used a fixed test set of size 1000. The size of training set varied from 2 to 50 in RG1 and from 4 to 100 in RG2.

4.1.2 Results and discussion

As shown by Fig. 9(a), all three systems have predictive accuracies significantly higher than default. Therefore Null hypothesis 1.1 is refuted. MC-TopLog is not usually able to carry out predicate invention, but is enabled to do so by the inclusion of a meta-interpreter as background knowledge.

As shown in Fig. 9(b), MC-TopLog's running time is considerably longer than Metagol$_R$ and ASP$_{\mathrm{M}R}$. MC-TopLog has slightly lower predictive accuracies than both Metagol$_R$ and ASP$_{\mathrm{M}R}$. The difference is statistically significant according to a t-test ($p < 0.01$). Therefore, Null hypothesis 1.2 is refuted with respect to both predictive accuracy and running time. MC-TopLog's longer running time is due to the fact that it enumerates all candidate hypotheses within the version space. By contrast, both Metagol$_R$ and ASP$_{\mathrm{M}R}$ do not traverse the entire space. In particular, ASP solver like Clasp incorporate effective optimisation techniques based on branch-and-bound algorithms (Gebser et al. 2007). The larger hypothesis space leads to lower accuracy in MC-TopLog. This is consistent with the Blumer Bound (Blumer et al. 1989), according to which the error bound decreases with the size of the hypothesis space. Moreover, MC-TopLog's accuracy is also affected by its covering algorithm which is greedy and does not guarantee finding a global optimal. By contrast, both Metagol$_R$ and ASP$_{\mathrm{M}R}$ find an hypothesis which is minimal in terms of description length. Figure 11 compares the different hypothesis suggested by the three systems. MC-TopLog's hypothesis $H_{mcTopLog}$ is longer than both of Metagol$_R$ and ASP$_{\mathrm{M}R}$. By contrast, both Metagol$_R$ and ASP$_{\mathrm{M}R}$ derive the one with minimal description length, although they are not exactly the same. $H_{metagolR}$ is more specific than H_{aspMR} due to the specific-to-general search in Metagol$_R$. In this example, $H_{metagolR}$ is the same as the target hypothesis.

[11] The running times of Metagol$_R$ and ASP$_{\mathrm{M}R}$ are measured in terms of getting the first minimal hypothesis, rather than all minimal hypotheses.

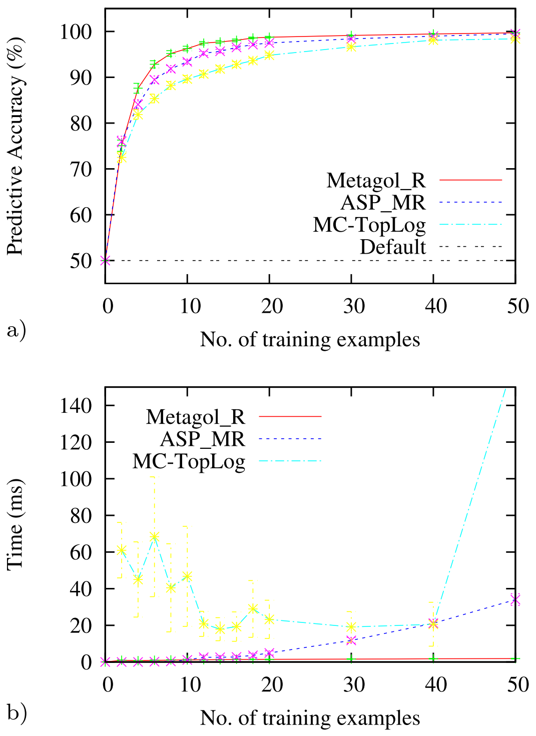


Fig. 9 Average (**a**) predictive accuracies and (**b**) running times for Null hypothesis 1 (Regular) on short sequence examples (RG1)

Figure 9(b) indicates that $\mathrm{Metagol}_R$ has considerably lower running time than $\mathrm{ASP}_{\mathrm{M}R}$, and the difference increases when examples are long, as shown in Fig. 10(b). $\mathrm{Metagol}_R$ also has slightly higher accuracy than $\mathrm{ASP}_{\mathrm{M}R}$. A t-test suggests that their difference in accuracy is statistically significant ($p < 0.01$) as one is consistently higher than the other. Therefore, Null hypothesis 1.3 is refuted with respect to both predictive accuracy and running time. The reasons that $\mathrm{Metagol}_R$ is faster than $\mathrm{ASP}_{\mathrm{M}R}$ on learning regular languages are: (1) $\mathrm{Metagol}_R$, as a Prolog implementation, can use forms of procedural bias which cannot be defined declaratively in ASP since the search in ASP is not affected by the order of clauses in the logic program; (2) there are few constraints in the learning task so that efficient constraint handling techniques in ASP do not increase efficiency.

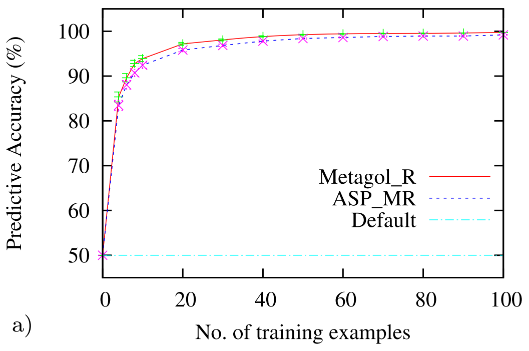
Both $\mathrm{Metagol}_R$ and $\mathrm{ASP}_{\mathrm{M}R}$'s running times appear to increase linearly with the number of examples. By contrast, MC-TopLog's running time appears to be unaffected by the number of examples. MC-TopLog's running time is determined by the size of the hypothesis space it enumerates, which depends on the lengths of examples. It therefore fails to learn from RG2 which has longer sequences (see Table 1).

4.2 Learning context-free languages

We investigate the following Null hypotheses.

Null Hypothesis 2.1 $\mathrm{Metagol}_{CF}$, $\mathrm{ASP}_{\mathrm{M}CF}$ and a state-of-the-art ILP system cannot learn randomly chosen Context-Free languages.

Null Hypothesis 2.2 $\mathrm{Metagol}_{CF}$ and $\mathrm{ASP}_{\mathrm{M}CF}$ cannot outperform a state-of-the-art ILP system on learning randomly chosen Context-Free languages.



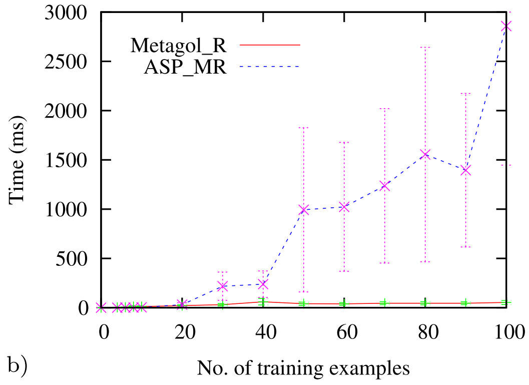


Fig. 10 Average (**a**) predictive accuracies and (**b**) running times for Null hypothesis 1 (Regular) on long sequence examples (RG2)

Null Hypothesis 2.3 Metagol$_{CF}$ cannot outperform ASP$_{\mathrm{MCF}}$ on learning randomly chosen Context-Free languages.

4.2.1 Materials and methods

Randomly chosen Context-Free grammars were generated using an SLP and reduced using Plotkin's reduction algorithm (see Sect. 2.5). Grammars were removed if they corresponded to finite languages or could be recognised using the pumping lemma for Context-Free grammars. However, not all Regular grammars can be filtered in this way, since it is undecidable whether a Context-Free grammar is Regular. More specifically, if a grammar is not pumpable, then it is definitely Regular, while a pumpable grammar is not necessarily non-Regular.

E^+	E^-	$H_{metagolR}$	H_{aspMR}	$H_{mcTopLog}$
aa aba $abbba$	$abab$ $aabaa$ $baaababaa$	$s \rightarrow a\ s_1$ $s_1 \rightarrow b\ s_1$ $s_1 \rightarrow a\ s_2$ $s_2 \rightarrow$	$s \rightarrow a\ s_1$ $s_1 \rightarrow b\ s_1$ $s_1 \rightarrow a\ \mathrm{s}$ $\mathrm{s} \rightarrow$	$s \rightarrow a\ s$ $s \rightarrow$ $s \rightarrow b\ s_1$ $s_1 \rightarrow a\ s_2$ $s_2 \rightarrow$ $s_1 \rightarrow b\ s$

Fig. 11 Hypothesis comparison

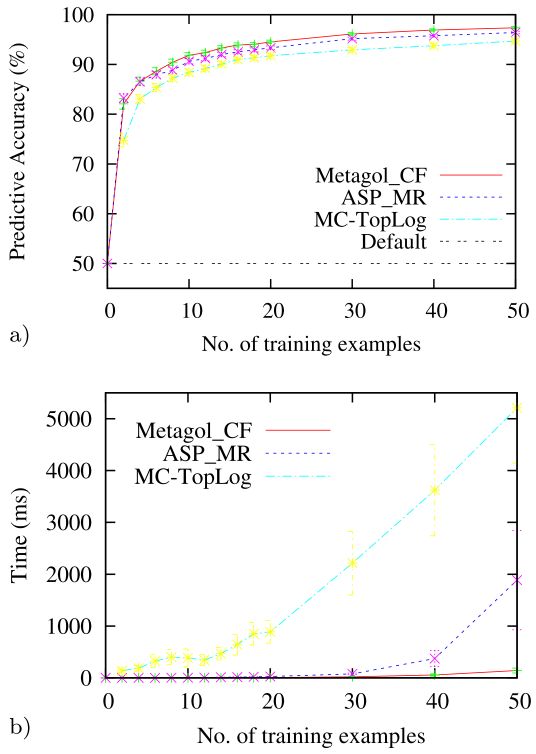


Fig. 12 Average (**a**) predictive accuracies and (**b**) running times for Null hypothesis 2 (Context-free) on short sequence examples (CFG3)

The examples were generated in the same way as that in the Regular-language experiment. There were two datasets, each containing 200 samples. Details are shown in Table 1. The comparisons of Metagol$_{CF}$, ASP$_{\mathrm{M}CF}$ and MC-TopLog on learning Context-Free grammars was done using only dataset CFG3 since MC-TopLog failed to terminate on CFG4 with long-sequence examples. The evaluation method was the same as that for learning regular languages.

4.2.2 Results and discussion

As shown in Fig. 12(a), all three systems derive hypotheses with predictive accuracies considerably higher than default. Therefore Null hypotheses 2.1 is refuted. Compared to MC-TopLog, both Metagol$_{CF}$ and ASP$_{\mathrm{M}CF}$ have consistently higher averaged predictive accuracies. This is again explained by the Blumer Bound since MC-TopLog considers a larger hypothesis space. Metagol$_{CF}$ conducts a bounded search using a bottom clause so that it is feasible even though the version space is potentially infinite. ASP solvers can also deal with infinite spaces.

Null hypothesis 2.2 is refuted with respect to both running time and predictive accuracy. The predictive accuracies of Metagol$_{CF}$ and ASP$_{\mathrm{M}CF}$, have no significant difference on either dataset, as shown by the graphs in Figs. 12(a) and 13(a), since both derive globally optimal solutions. However, Metagol$_{CF}$ has shorter running time due to its procedural bias. Therefore Null hypothesis 2.3 is refuted.

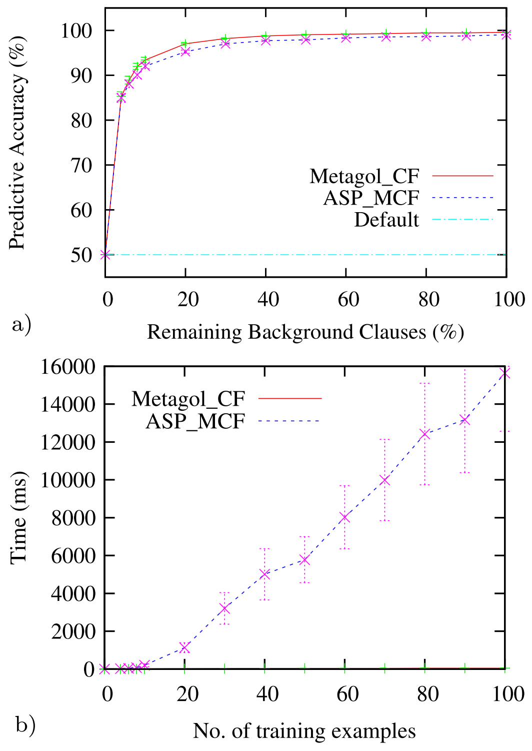


Fig. 13 Average (**a**) predictive accuracies and (**b**) running times for Null hypothesis 2 (Context-free) on long sequence examples (CFG4)

4.3 Representation change

Null Hypothesis 3 Metagol$_{RCF}$ cannot improve performance by changing representation from Regular to Context-Free languages.

4.3.1 Materials and methods

The experiment used RG1 and CFG3 from the previous two experiments. Therefore, there were 400 sampled grammars in total, half being Regular and the other half mostly Context-Free and non-Regular.

We compared Metagol$_{RCF}$ (variable hypothesis space) against Metagol$_{CF}$ (fixed hypothesis space). The predictive accuracies and running time were measured as before. The results were averaged over the 400 grammars.

4.3.2 Results and discussion

As shown in Fig. 14(a), Metagol$_{RCF}$ has slightly higher predictive accuracies than Metagol$_{CF}$. This refutes Null hypothesis 3. The accuracy difference is once more consistent with the Blumer Bound (Blumer et al. 1989), according to which the error bound decreases with the size of the hypothesis space.

Note also in Fig. 14(b), that the running times of Metagol$_{CF}$ are significantly higher than Metagol$_{RCF}$. This can be explained by the fact that when the target grammar is Regular, Context-Free grammars were still considered.

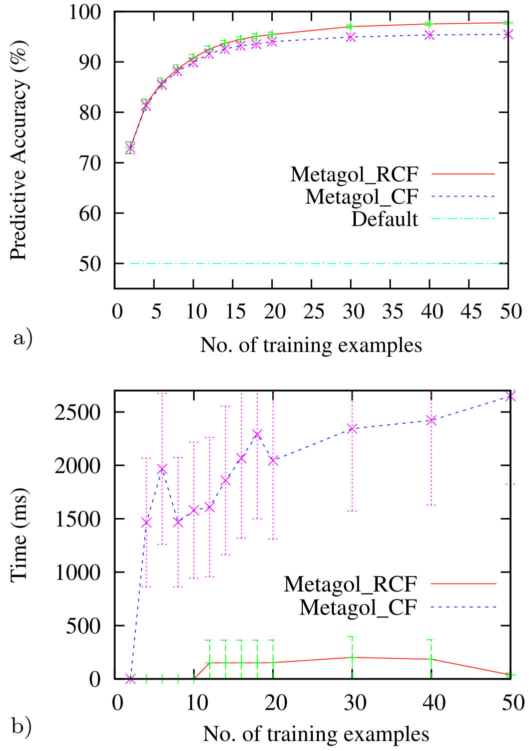


Fig. 14 Average (**a**) predictive accuracies and (**b**) running times for Null hypothesis 3 (Representation change) on combination of RG dataset1 and CFG dataset3

4.4 Learning a simplified natural language grammar

Metagol$_N$ and ASP$_{MN}$ are two systems resulting from the application of Metagol and ASP$_M$ in learning a simplified natural language grammar. We investigate the following Null hypotheses. MC-TopLog was not included for comparison since its search time was excessive in these learning tasks.

Null Hypothesis 4.1 Metagol$_N$ and ASP$_{MN}$ cannot learn a simplified natural language grammar.

Null Hypothesis 4.2 Metagol$_N$ cannot outperform ASP$_{MN}$ on learning a simplified natural language grammar.

Null Hypothesis 4.3 The provision of background knowledge does not improve learning accuracies and efficiency.

4.4.1 Materials and methods

The training examples come from the same domain considered in Muggleton et al. (2012) and consist of 50 sentences such as "a ball hits the small dog". Half the examples are positive and half negative, resulting in a default accuracy of 50 %. The complete target grammar rules for parsing the training examples are given in Fig. 15. Each learning task is generated by randomly removing a set of clauses. The left-out clauses become the target to be reconstructed. For each size of leave-out, we sampled ten times. For each sample, the predictive

Definite Clause Grammar	**Production rules**
$s(S1, S2) \leftarrow np(S1, S3), vp(S3, S4), np(S4, S2).$	$s \rightarrow s4\ s_1$
$s(S1, S2) \leftarrow np(S1, S3), vp(S3, S4), np(S4, S5),$	$s_1 \rightarrow s5\ s4$
$prep(S5, S6), np(S6, S2).$	$s_1 \rightarrow s5\ s2$
	$s2 \rightarrow s4\ s3$
	$s3 \rightarrow prep\ s4$
$np(S1, S2) \leftarrow det(S1, S3), noun(S3, S2).$	$s4 \rightarrow det\ noun$
$np(S1, S2) \leftarrow det(S1, S3), adj(S3, S4), noun(S4, S2).$	$s4 \rightarrow det\ s6$
	$s6 \rightarrow adj\ noun$
$vp(S1, S2) \leftarrow verb(S1, S2).$	$s5 \rightarrow verb$
$vp(S1, S2) \leftarrow verb(S1, S3), prep(S3, S2).$	$s5 \rightarrow verb\ prep$

Fig. 15 Target theory for simplified natural language grammar

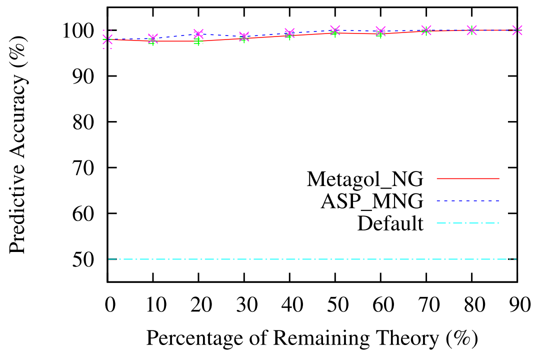


Fig. 16 Average predictive accuracies for Null hypothesis 4 on simplified natural language grammars

accuracies were computed by 10-fold cross validation.[12] The results plotted on the figure are averaged over all leave-out samples.

4.4.2 Results and discussion

The predictive accuracies and running times are plotted in Figs. 16 and 17 respectively. The x-axis corresponds to the percentage of remaining production rules. More specifically, 0 % corresponds to the case when $B_A = \emptyset$, while 90 % means 9 out of 10 production rules remain. Figure 16 shows that the predictive accuracies of both Metagol$_N$ and ASP$_{\mathrm{M}N}$ are significantly higher than default, therefore Null hypothesis 4.1 is refuted.

Although there is no significant difference between Metagol$_N$ and ASP$_{\mathrm{M}N}$ in terms of predictive accuracy, ASP$_{\mathrm{M}N}$ takes much shorter running time than Metagol$_N$ when more than half of the production rules are missing ($x < 50$ %). However, the expanded version for 50 % $\leq x \leq$ 90 % in Fig. 17(b) shows that ASP$_{\mathrm{M}N}$ becomes slower than Metagol$_N$ when background knowledge is less sparse. Therefore, Null hypothesis 4.2 is refuted since when more than 70 % of the production rules remain Metagol$_N$ has significantly shorter running time than ASP$_{\mathrm{M}N}$ without sacrificing its predictive accuracy. This is due to the procedural bias encoded in Metagol$_N$.

The running times of both Metagol$_N$ and ASP$_{\mathrm{M}N}$ decrease dramatically with the increase of background knowledge. The predictive accuracies increase with increasing background

[12]The size of available examples is 50, therefore not large enough for reserving a subset as test set.

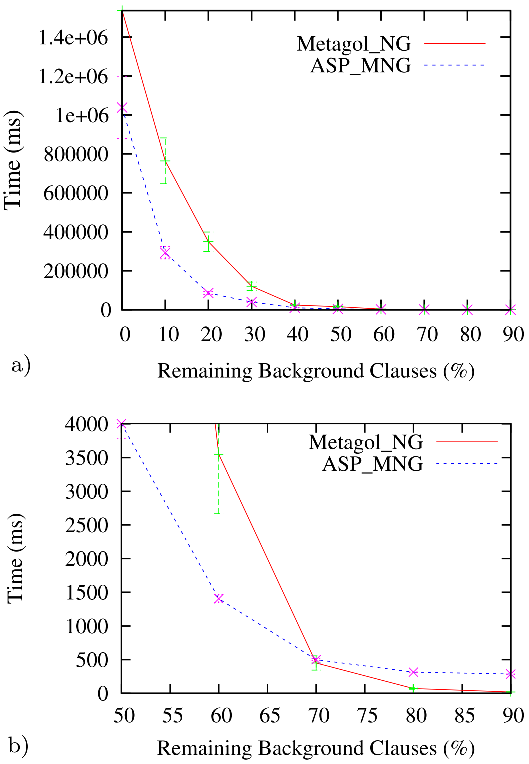


Fig. 17 Averaged running time for Null hypothesis 4 on simplified natural language grammars. (**a**) Full range [0, 90]. (**b**) Partial range [50, 90] but expanded

Fig. 18 Metagol and ASP_M hypotheses for learning a simplified natural language grammar

H_M(Metagol)	H_A (ASP_M)
$s \rightarrow s_2\ s_1$	$s \rightarrow s_4\ s_3$
$s_1 \rightarrow prep\ s_4$	$s_3 \rightarrow verb\ s_5$
$s_1 \rightarrow det\ s_5$	$s_3 \rightarrow verb\ s_4$
$s_2 \rightarrow s_4\ s_3$	$s_4 \rightarrow s_6\ s_5$
$s_3 \rightarrow verb$	$s_4 \rightarrow det\ s_5$
$s_3 \rightarrow verb\ s_4$	$s_5 \rightarrow prep\ s_6$
$s_4 \rightarrow det\ s_5$	$s_5 \rightarrow noun$
$s_5 \rightarrow adj\ noun$	$s_6 \rightarrow det\ noun$
$s_5 \rightarrow noun$	$s_6 \rightarrow det\ adj$

knowledge, reaching 100 % when the degree of remaining background clauses increases to 70 %. Therefore, Null hypothesis 4.3 is refuted.

Figure 18 compares the different hypotheses derived by Metagol_N and $\text{ASP}_{\text{M}N}$. These are derived when $B_A = \emptyset$. Since both Metagol_N and $\text{ASP}_{\text{M}N}$ find an hypothesis which is globally optimal in terms of description length, these hypotheses have identical description length although they are not identical hypotheses. Among all the invented predicates in H_M, s_4 corresponds to np in natural grammars and s_3 is closed to vp. Similarly in H_A, s_3 and $s6$ corresponds to vp and np respectively.

$$\begin{aligned} staircase(Planes) \leftarrow\ & n\_of\_parts(Planes, 4), \text{ \% there are 4 parts in } Planes \\ & member(C, Planes),\ distributed\_along(C, axisX). \end{aligned}$$

Fig. 19 Non-recursive definition of staircase hypothesised by ALEPH (Partial)

First-order logic	**Production rules**
$staircase(Planes) \leftarrow s_1(Planes).$	$staircase \rightarrow s_1$
$staircase([X, Y, Z \mid Planes]) \leftarrow s_1([X, Y, Z]),$ $staircase([Z \mid Planes]).$	$staircase \rightarrow s_1\ staircase$
$s_1([X, Y, Z]) \leftarrow vertical(X, Z), horizontal(Z, Y)$	$s_1 \rightarrow vertical\ horizontal$

Fig. 20 Recursive definition of staircase hypothesised by MIL. s_1 is an invented predicate corresponding to the concept of *step*

4.5 Learning a definition of a staircase

The authors of Farid and Sammut (2012) have shown that ALEPH can learn a definition of a staircase for a rescue robot from visually-derived data. Part of such definition is shown in Fig. 19. This kind of definition is not entirely general since it does not involve recursion. We now demonstrate that MIL can be used to learn a general recursive definition of a staircase using predicate invention. A staircase can be represented by a set of ordered planes. For example, $staircase([p1, p2, p3])$ represents a staircase composed of three planes. Relational information from the camera indicates that $plane1$ is vertical relative to $plane2$. This can be encoded as a delta rule $delta4(vertical, p1, p2)$, where $vertical$ is a non-terminal of a grammar and $p1$ and $p2$ are terminals. The meta-interpreter used in this experiment is a variant of the Context-Free Meta-interpreter from Fig. 3.

Training examples of staircases and their planar description were provided as input to both Metagol and ASP_M. The resulting hypothesis produced by both systems is shown in Fig. 20, where s_1 is an invented predicate corresponding to *step*. Due to its recursive form, this definition has shorter description length than those found by ALEPH. It is also general in its applicability and easily understood.

5 Related work

Grammatical inference (or grammatical induction) is the process of learning a grammar from a set of examples. It is closely related to the fields of machine learning as well as the theory of formal languages. It has numerous real-world applications including speech recognition (e.g. Stolcke 1995), computational linguistics (e.g. Florêncio 2002) and computational biology (e.g. Salvador and Benedi 2002).

The problem of learning or inferring Regular languages, which can be represented by deterministic finite state automata, has been well studied and efficient automaton-based learning algorithms have existed since the 1950s (Moore 1956). Some heuristic approaches to machine learning context-free grammars (Vanlehn and Ball 1987; Langley and Stromsten 2000) have been investigated, though the completeness of these approaches is unclear. Although an efficient and complete approach exists for learning context-free grammars from parse trees (Sakakibara 1992), no comparable complete approach exists in the literature for learning context-free grammars from positive and negative samples of the language. According to a recent survey article learning context-free languages is widely believed to be intractable and the state of the art mainly consists of negative results (de la Higuera 2005).

There are some positive PAC (probably approximately correct) learning results concerning Regular languages (e.g. Denis 2001), but to the best of our knowledge, these have not been extended to the context-free case. The difficulty of learning context-free languages arises from a very large search space compared to regular languages.

ILP, among other learning methods, has previously been applied to grammatical inference (e.g. Boström 1998). However, as discussed in Sect. 1, ILP systems normally require predicate invention even for learning Regular languages. Predicate invention has been viewed as an important problem since the early days of ILP (e.g. Muggleton and Buntine 1988), but it is widely accepted to be a hard and under-explored topic within ILP (Muggleton et al. 2011). Although Cussens and Pulman (2000) has applied ALEPH for learning natural language grammar, its learning setting avoids predicate invention by assuming all predicates like *np* (noun phrase) are known in the background knowledge. Additionally, the entailment-incompleteness of ALEPH restricts the applicability of the approach.

In the Meta-interpretive Learning (MIL) framework introduced in this paper, predicate invention is done via abduction with respect to a meta-interpreter and by the introduction of first-order variables. This method is therefore related to other studies where abduction has been used for predicate invention. For instance, (Inoue et al. 2010) assumes background knowledge such as the following.

$$caused(X, Y) \leftarrow connected(X, Y).$$

$$caused(X, Y) \leftarrow connected(X, Z), caused(Z, Y).$$

Here the predicates *connected* and *caused* are both *meta-predicates* for object-level propositions g and s. Given multiple observations such as $caused(g, s)$ and $caused(h, s)$ abduction can be used to generate an explanation

$$\exists X\big(connected(g, X), connected(h, X), connected(X, s)\big)$$

in which X can be thought of as a new propositional predicate. One important feature of MIL, which makes it distinct from this approach, is that it introduces new predicate symbols which represent relations rather than new objects or propositions. In comparison to previous approaches to predicate invention one might question what is meant by the predicate symbols being *new*. In our case, we assume a source containing either a finite or an infinite source (e.g. the natural numbers) of uninterpreted predicate symbols. Rather than providing these implicitly in hidden code (as was the case in CIGOL (Muggleton and Buntine 1988)), we prefer to have these symbols explicitly defined as part of the Herbrand universe of the meta-interpreter. Abductive hypothesis formation then provides the interpretation for these otherwise uninterpreted symbols.

6 Conclusions and further work

This paper explores the theory, implementation and experimental application of a new framework (MIL) for machine learning by abduction with respect to a given Meta-interpreter. We have demonstrated that the MIL framework can be implemented using a simple Prolog program or within a more sophisticated solver such as ASP. We have applied these implementations to the problem of inductive inference of grammars, where our experiments indicate that they compete favourably in speed and accuracy with the state of the art ILP system MC-TopLog. The MIL framework has a number of advantages with respect to the standard ILP framework. In particular, predicate invention and mutual recursion can be incorporated with ease by way of Skolem constants. The Meta-interpreter provides an efficient declarative bias mechanism for controlling the search for hypotheses, which takes advantage of the completeness of SLD resolution in Prolog. This mechanism is distinct from the use of first-order declarative bias in the form of a $\top$ theory (Muggleton et al. 2010, 2012) since it is not assumed that the meta-interpreter entails each hypothesis.

The approach presented here is limited to learning grammars in the form of DCGs. Such grammars can be learned with predicates of arity at most 2. In future work we hope to deal with a number of extensions of this study. In particular, we would like to extend the applications of the MIL framework to non-grammar fragments of first-order logic. We have shown an example of non-grammar learning, but more general learning problem requires Monadic and Dyadic and higher arity fragments of first-order logic. We would like to incorporate a number of other features of ILP and SRL learning systems such as probabilistic parameters (similar to SRL) and noise handling.

Clearly devising an appropriate meta-interpreter for a fragment of logic other than those studied in this paper will require careful mathematical analysis. The situation may be compared to that within Support Vector Machines, in which certain mathematical properties have to be established for each new form of kernel function. Hopefully, over time, such a process will become more routine and it may be possible to provide end users with general tools which support this activity. In the ideal case, we would like in future work, to develop a meta-interpreter which is capable of implementing highly expressive, ideally Turing-complete, languages. Such a meta-interpreter might then be reasonably expected to learn effectively on arbitrary new problems without further manual revision of its meta-interpreter.

In closing we believe the MIL framework provides a promising and novel form of Inductive Logic Programming which avoids a number of the bottlenecks of existing approaches.

References

Andres, B., Kaufmann, B., Matheis, O., & Schaub, T. (2012). Unsatisfiability-based optimization in clasp. In *Proceedings of the 28th International Conference on Logic Programming*.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, *36*(4), 929–965.

Boström, H. (1998). Predicate invention and learning from positive examples only. In *10th European Conference on Machine Learning (ECML-98)* (pp. 226–237). Berlin: Springer.

Cussens, J., & Pulman, S. (2000). Experiments in inductive chart parsing. In J. Cussens & S. Dzeroski (Eds.), *LNAI: Vol. 1925. Proceedings of Learning Language in Logic (LLL2000)* (pp. 143–156). Berlin: Springer.

de la Higuera, C. (2005). A bibliographical study of grammatical inference. *Pattern Recognition*, *38*, 1332–1348.

Denis, F. (2001). Learning regular languages from simple positive examples. *Machine Learning*, *44*(1), 37–66.

Farid, R., & Sammut, C. (2012, to appear). Plane-based object categorization using relational learning. ILP2012 MLJ special issue.

Flach, P. A. & Kakas, A. C. (Eds.) (2000). *Abductive and Inductive Reasoning*. *Pure and Applied Logic.* Amsterdam: Kluwer.

Florêncio, C. (2002). Consistent identification in the limit of rigid grammars from strings is np-hard. In *Grammatical Inference: Algorithms and Applications* (pp. 729–733).

Gebser, M., Kaminski, R., Kaufmann, B., & Schaub, T. (2012). *Answer Set Solving in Practice*. *Synthesis Lectures on Artificial Intelligence and Machine Learning*. San Mateo: Morgan and Claypool.

Gebser, M., Kaufmann, B., Neumann, A., & Schaub, T. (2007). clasp: A conflict-driven answer set solver. In C. Baral, G. Brewka, & J. Schlipf (Eds.), *Lecture Notes in Computer Science: Vol. 4483*. *Logic Programming and Nonmonotonic Reasoning* (pp. 260–265). Berlin: Springer.

Hopcroft, J. E., & Ullman, J. D. (1979). *Introduction to Automata and Formal Languages*. Reading: Addison-Wesley.

Inoue, K., Furukawa, K., Kobayashiand, I., & Nabeshima, H. (2010). Discovering rules by meta-level abduction. In L. De Raedt (Ed.), *LNAI: Vol. 5989. Proceedings of the Nineteenth International Conference on Inductive Logic Programming (ILP09)* (pp. 49–64). Berlin: Springer.

Kakas, A. C., Van Nuffelen, B., & Denecker, M. (2001). A-system: Problem solving through abduction. In *IJCAI* (pp. 591–596).

Langley, P., & Stromsten, S. (2000). Learning context-free grammars with a simplicity bias. In R. López de Mántaras & E. Plaza (Eds.), *Lecture Notes in Computer Science: Vol. 1810. Machine Learning: ECML 2000* (pp. 220–228). Berlin: Springer.

Moore, E. F. (1956). Gedanken-experiments on sequential machines. In C. E. Shannon & J. McCarthy (Eds.), *Automata Studies* (pp. 129–153). Princeton: Princeton University Press.

Muggleton, S. H. (1990). *Inductive Acquisition of Expert Knowledge*. Wokingham: Addison-Wesley.

Muggleton, S. H. (1995). Inverse entailment and Progol. *New Generation Computing*, *13*, 245–286.

Muggleton, S. H. (1996). Stochastic logic programs. In L. de Raedt (Ed.), *Advances in Inductive Logic Programming* (pp. 254–264). Amsterdam: IOS Press.

Muggleton, S. H., & Bryant, C. H. (2000). Theory completion using inverse entailment. In *Proc. of the 10th International Workshop on Inductive Logic Programming (ILP-00)* (pp. 130–146). Berlin: Springer.

Muggleton, S. H., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning* (pp. 339–352). Los Altos: Kaufmann.

Muggleton, S. H., Lin, D., & Tamaddoni-Nezhad, A. (2012). MC-Toplog: Complete multi-clause learning guided by a top theory. In *LNAI: Vol. 7207. Proceedings of the 21st International Conference on Inductive Logic Programming* (pp. 238–254).

Muggleton, S. H., De Raedt, L., Poole, D., Bratko, I., Flach, P., & Inoue, K. (2011). ILP turns 20: biography and future challenges. *Machine Learning*, *86*(1), 3–23.

Muggleton, S. H., Santos, J., & Tamaddoni-Nezhad, A. (2010). TopLog: ILP using a logic program declarative bias. In *LNCS: Vol. 5366. Proceedings of the International Conference on Logic Programming 2008* (pp. 687–692). Berlin: Springer.

Muggleton, S. H., & Pahlavi, N. (2012, in press). Towards efficient higher-order logic learning in a first-order datalog framework. In *Latest Advances in Inductive Logic Programming*. London: Imperial College Press.

Nabeshima, H., Iwanuma, K., Inoue, K., & Ray, O. (2010). Solar: An automated deduction system for consequence finding. *AI Commun.*, *23*(2–3), 183–203.

Nienhuys-Cheng, S.-H., & de Wolf, R. (1997). *LNAI: Vol. 1228. Foundations of Inductive Logic Programming*. Berlin: Springer.

Plotkin, G. D. (1969). A note on inductive generalisation. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence* (Vol. 5, pp. 153–163). Edinburgh: Edinburgh University Press.

De Raedt, L. (2012). Declarative modeling for machine learning and data mining. In *Proceedings of the International Conference on Algorithmic Learning Theory* (p. 12).

Sakakibara, Y. (1992). Efficient learning of context-freegrammars from positive structural examples. *Information and Computation*, *97*(1), 23–60.

Salvador, I., & Benedi, J. M. (2002). Rna modeling by combining stochastic context-free grammars and n-gram models. *International Journal of Pattern Recognition and Artificial Intelligence*, *16*(3), 309–316.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, *21*(2), 165–201.

Vanlehn, K., & Ball, W. (1987). A version space approach to learning context-free grammars. *Machine Learning*, *2*, 39–74.