Mach Learn
DOI 10.1007/s10994-014-5471-y

Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited

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

Received: 10 January 2014 / Accepted: 10 October 2014

Abstract Since the late 1990s predicate invention has been under-explored within inductive logic programming due to difficulties in formulating efficient search mechanisms. However, a recent paper demonstrated that both predicate invention and the learning of recursion can be efficiently implemented for regular and context-free grammars, by way of metalogical substitutions with respect to a modified Prolog meta-interpreter which acts as the learning engine. New predicate symbols are introduced as constants representing existentially quantified higher-order variables. The approach demonstrates that predicate invention can be treated as a form of higher-order logical reasoning. In this paper we generalise the approach of meta-interpretive learning (MIL) to that of learning higher-order dyadic datalog programs. We show that with an infinite signature the higher-order dyadic datalog class H^2_2 has universal Turing expressivity though H^2_2 is decidable given a finite signature. Additionally we show that Knuth–Bendix ordering of the hypothesis space together with logarithmic clause bounding allows our MIL implementation Metagol$_D$ to PAC-learn minimal cardinality H^2_2 definitions. This result is consistent with our experiments which indicate that Metagol$_D$ efficiently learns compact H^2_2 definitions involving predicate invention for learning robotic strategies, the East–West train challenge and NELL. Additionally higher-order concepts were learned in the NELL language learning domain. The Metagol code and datasets described in this paper have been made publicly available on a website to allow reproduction of results in this paper.

Keywords Induction · Abduction · Meta-interpretation · Predicate invention · Learning recursion

Editors: Gerson Zaverucha and Vítor Santos Costa.

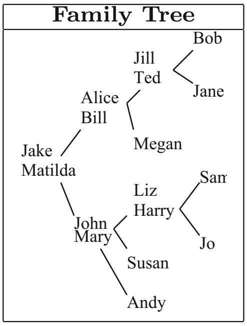
S. H. Muggleton (✉) · D. Lin · A. Tamaddoni-Nezhad
Department of Computing, Imperial College London, London, UK
e-mail: s.muggleton@imperial.ac.uk

Published online: 12 March 2015

1 Introduction

Suppose we machine learn a set of kinship relations such as those in Fig. 1. If examples of the ancestor relation are provided and the background contains only father and mother facts, then a system must not only be able to learn ancestor as a recursive definition but also simultaneously *invent* parent to learn these definitions.

Although the topic of Predicate Invention was investigated in early Inductive Logic Programming (ILP) research (Muggleton and Buntine 1988; Stahl 1992) it is still seen as hard and under-explored (Muggleton et al. 2011). ILP systems such as ALEPH (Srinivasan 2001) and FOIL (Quinlan 1990) have no predicate invention and limited recursion learning and therefore cannot learn recursive grammars from example sequences. By contrast, in (Muggleton et al. 2014) definite clause grammars were learned with predicate invention using Meta-Interpretive Learning (MIL). MIL (Muggleton and Lin 2013; Muggleton et al. 2014; Lin et al. 2014) is a technique which supports efficient predicate invention and learning of recursive logic programs built as a set of metalogical substitutions by a modified Prolog meta-interpreter (see Fig. 2) which acts as the central part of the ILP learning engine. The meta-interpreter is provided by the user with *meta-rules* (see Fig. 3) which are higher-order expressions describing the forms of clauses permitted in hypothesised programs. As shown in Fig. 3 each meta-rule has an associated Order constraint, which is designed to ensure termination of the proof (see Sect. 4.1). The meta-interpreter attempts to prove the examples and, for any successful proof, saves the substitutions for existentially quantified variables found in the associated meta-rules. When these substitutions are applied to the meta-rules


Family Tree

Target Theory
$father(ted, bob) \leftarrow$ $father(ted, jane) \leftarrow$ $parent(X, Y) \leftarrow mother(X, Y)$ $parent(X, Y) \leftarrow father(X, Y)$ $ancestor(X, Y) \leftarrow parent(X, Y)$ $ancestor(X, Y) \leftarrow parent(X, Z), ancestor(Z, Y)$

First-order	Metalogical substitutions
Examples $ancestor(jake, bob) \leftarrow$ $ancestor(alice, jane) \leftarrow$	N/A
Background Knowledge $father(jake, alice) \leftarrow$ $mother(alice, ted) \leftarrow$	N/A
Instantiated Hypothesis $father(ted, bob) \leftarrow$ $father(ted, jane) \leftarrow$ $p1(X, Y) \leftarrow father(X, Y)$ $p1(X, Y) \leftarrow mother(X, Y)$ $ancestor(X, Y) \leftarrow p1(X, Y)$ $ancestor(X, Y) \leftarrow p1(X, Z), ancestor(Z, Y)$	$metasub(instance, [father, ted, bob])$ $metasub(instance, [father, ted, jane])$ $metasub(base, [p1, father])$ $metasub(base, [p1, mother])$ $metasub(base, [ancestor, p1])$ $metasub(tailrec, [ancestor, p1, ancestor])$

Fig. 1 Kinship example. *p1* invented, representing *parent*

```
Generalised meta-interpreter
  prove([], Prog, Prog).
  prove([Atom|As], Prog1, Prog2) : −
      metarule(Name, MetaSub, (Atom :- Body), Order),
      Order,
      save_subst(metasub(Name, MetaSub), Prog1, Prog3),
      prove(Body, Prog3, Prog4),
      prove(As, Prog4, Prog2).
```

Fig. 2 Prolog code for the generalised meta-interpreter. The interpreter recursively proves a series of atomic goals by matching them against the heads of meta-rules. After testing the Order constraint *save_subst* checks whether the meta-substitution is already in the program and otherwise adds it to form an augmented program. On completion the returned program, by construction, derives all the examples

Name	Meta-Rule	Order
Instance	$P(X, Y) \leftarrow$	$True$
Base	$P(x, y) \leftarrow Q(x, y)$	$P \succ Q$
Chain	$P(x, y) \leftarrow Q(x, z), R(z, y)$	$P \succ Q, P \succ R$
TailRec	$P(x, y) \leftarrow Q(x, z), P(z, y)$	$P \succ Q,$ $x \succ z \succ y$

Fig. 3 Examples of dyadic meta-rules with associated Herbrand ordering constraints. $\succ$ is a pre-defined ordering over symbols in the signature

they result in a first-order definite program which is an inductive generalisation of the examples. For instance, the two examples shown in the upper part of Fig. 1 could be proved by the meta-interpreter in Fig. 2 from the Background Knowledge BK by generating the Hypothesis H using the Prolog goal

$$\leftarrow prove([ancestor, jake, bob], [ancestor, alice, jane], BK, H).$$

H is constructed by applying the metalogical substitutions in Fig. 1 to the corresponding meta-rules found in Fig. 3. Note that $p1$ is an invented predicate corresponding to *parent*.

Completeness of SLD resolution ensures that *all* hypotheses consistent with the examples can be constructed. Moreover, unlike many ILP systems, *only* hypotheses consistent with all examples are considered. Owing to the efficiency of Prolog backtracking MIL implementations have been demonstrated to search the hypothesis space 100–1,000 times faster than state-of-the-art ILP systems (Muggleton et al. 2014) in the task of learning recursive grammars.[1] In this paper we investigate MIL's efficiency and completeness with respect to the broader class of Dyadic Datalog programs. We show that a fragment of this class is Turing equivalent, allowing the learning of complex recursive programs such as robot strategies.

1.1 Organisation of paper

The paper is organised as follows. In Sect. 2 we provide a comparison to related work. Section 3 describes the MIL framework. The implementation of the Metagol$_D$[2] system is then given in Sect. 4. Experiments on predicate invention and recursion for 1) structuring robot strategies, 2) the East–West trains competition data and 3) construction of concepts for the NELL language learning domain are given in Sect. 5 together with a reference to the website from which the Metagol code and datasets can be obtained. Lastly we conclude the paper and discuss future work in Sect. 6.

[1] Metagol$_R$ and Metagol$_{CF}$ learn Regular and Context-Free grammars respectively.

[2] Metagol$_D$ learns Dyadic Datalog programs.

2 Related work

Predicate Invention has been viewed as an important problem since the early days of ILP (e.g. Muggleton and Buntine 1988; Rouveirol and Puget 1989; Stahl 1992) since it is essential for automating the introduction of auxiliary predicates within top-down programming. Early approaches were based on the use of W operators within the inverting resolution framework (Muggleton and Buntine 1988; Rouveirol and Puget 1989). However, apart from the inherent problems in controlling the search, the completeness of these approaches was never demonstrated, partly because of the lack of a declarative bias to delimit the hypothesis space. This led to particular limitations such as the approaches being limited to introducing a single new predicate call as the tail literal of the calling clause. Failure to address these issues has led to limited progress being made in this important topic over a protracted period (Muggleton et al. 2011). In the MIL framework described in Muggleton et al. (2014) and in this paper, predicate invention is conducted via construction of substitutions for meta-rules employed by a meta-interpreter. The use of the meta-rules clarifies the declarative bias being employed. New predicate names are introduced as higher-order skolem constants, a finite number of which are introduced during every iterative deepening of the search.

MIL is related to other studies where abduction has been used for predicate invention (e.g. Inoue et al. 2010). One important feature of MIL, which distinguishes it from other existing approaches, is that it introduces new predicate symbols which represent relations rather than new objects or propositions. This is critical for challenging applications such as robot planning. The NELL language learning task (Sect. 5.3) separately demonstrates MIL's abilities for learning higher-order concepts such as symmetry.

By comparison with other forms of declarative bias in ILP, such as modes (Muggleton 1995; Srinivasan 2001) or grammars (Cohen 1994), meta-rules are logical statements. This provides the potential for reasoning about them and manipulating them alongside normal first-order background knowledge. For instance, in Cropper and Muggleton (2014) it is demonstrated that sets of irreducible, or minimal sets of meta-rules can be found automatically by applying Plotkin's clausal theory reduction algorithm to an enumeration of all meta-rules in a given finite hypothesis language, resulting in meta-rules which exhibit lower runtimes and higher predictive accuracies. Moreover logical equivalence with the larger set ensures completeness of the reduced set.

The use of proof-completion in MIL is in some ways comparable to that used in Explanation-Based Generalisation (EBG) (De Jong 1981; Kedar-Cabelli and McCarty 1987; Mitchell et al. 1986). In EBG the proof of an example leads to a specialisation of the given domain theory leading to the generation of a special-purpose sub-theory described within a user-defined operational language. By contrast, in MIL the derivation of the examples is made from a higher-order program, and results in a first-order program based on a set of substitutions into the higher-order variables. A key difference is that inductive generalisation and predicate invention are not achieved in existing EBG paradigms, which assume a complete first-order domain theory. By contrast, induction, abduction and predicate invention are all achieved in MIL by way of the meta-rules. Owing to the existentially quantified variables in the meta-rules, the resulting first-order theories are strictly logical generalisation of the meta-rules. Viewed from the perspective of Formal Methods, the meta-rules in MIL can be viewed as a *Program Specification* in the style advocated by Hoare (1992; Hoare and Jifeng 2001).

Although McCarthy (1999) long advocated the use of higher-order logic for representing common sense reasoning, most knowledge representation languages avoid higher-order quantification owing to problems with decidability (Huet 1975) and theorem-proving efficiency. λ-Prolog (Miller 1991), is a notable exception which achieves efficient unification through assumptions on the flexibility of terms. Various authors (Feng and Muggleton 1992; Lloyd 2003) have advocated higher-order logic learning frameworks. However, to date these approaches have difficulties in incorporation of background knowledge and compatibility with more developed logic programming frameworks. Second-order logic clauses are used by Davis and Domingo (2009) as templates for rule learning in Markov logic. Here tractability is achieved by avoiding theorem proving. Higher-order rules have also been encoded in first-order Markov Logic (Sorower et al. 2011) to bias learning in tasks involving knowledge extraction from text. As a knowledge representation, higher-order datalog (see Sect. 3), first introduced in Pahlavi and Muggleton (2012), has advantages in being both expressive and decidable. The NELL language application (Sect. 5.3) demonstrates that higher-order concepts can be readily and naturally expressed in H^2_2 and learned within the MIL framework.

Relational Reinforcement (Džeroski et al. 2001) has been used in the context of learning robot strategies (Katz et al. 2008; Driessens and Ramon 2003). However, unlike the approach for learning recursive robot strategies described in Sect. 5.1 the Relational Reinforcement approaches are based on the use of ground instances and do not involve predicate invention to extend the relational vocabulary provided by the user. The approach of Pasula and Lang (Pasula et al. 2004) for learning STRIPS-like operators in a relational language is closer to the approach described in Sect. 5.1, but is restricted to learning non-recursive operators and does not involve predicate invention.

3 MIL framework

3.1 Logical notation

A variable is represented by an upper case letter followed by a string of lower case letters and digits. A function symbol is a lower case letter followed by a string of lower case letters and digits. A predicate symbol is a lower case letter followed by a string of lower case letters and digits. The set of all predicate symbols is referred to as the predicate signature and denoted $\mathcal{P}$. An arbitrary reference total ordering over the predicate signature is denoted $\succeq_{\mathcal{P}}$. The arity of a function or predicate symbol is the number of arguments it takes. A constant is a function or predicate symbol with arity zero. The set of all constants is referred to as the constant signature and denoted $\mathcal{C}$. An arbitrary reference total ordering over the constant signature is denoted $\succeq_{\mathcal{C}}$. Functions and predicate symbols are said to be monadic when they have arity one and dyadic when they have arity two. Variables and constants are terms, and a function symbol immediately followed by a bracketed n-tuple of terms is a term. A variable is first-order if it can be substituted for by a term. A variable is higher-order if it can be substituted for by a predicate symbol. A predicate symbol or higher-order variable immediately followed by a bracketed n-tuple of terms is called an atomic formula or atom for short. The negation symbol is $\neg$. Both A and $\neg A$ are literals whenever A is an atom. 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

Name	Meta-Rule	Quantified version
Instance	$P(X,Y) \leftarrow$	$\exists PXY\ P(X,Y)$
Base	$P(x,y) \leftarrow Q(x,y)$	$\exists PQ\forall xy\ P(x,y) \leftarrow Q(x,y)$
Chain	$P(x,y) \leftarrow Q(x,z), R(z,y)$	$\exists PQR\forall xyz\ P(x,y) \leftarrow Q(x,z), R(z,y)$
TailRec	$P(x,y) \leftarrow Q(x,z), P(z,y)$	$\exists PQ\forall xyz\ P(x,y) \leftarrow Q\ (x,z), P(z,y)$

Fig. 4 Quantification of meta-rules in Fig. 3

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 $(C_1 \wedge C_2 \wedge \ldots)$. 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 logic program. A logic program in which each clause is definite is called a definite program. 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 and σ, τ be sets of variables. $\exists\sigma.E$ and $\forall\tau.E$ are wffs. E is said to be ground whenever it contains no variables. E is said to be higher-order whenever it contains at least one higher-order variable or a predicate symbol as an argument of a term. E is said to be datalog if it contains no function symbols other than constants. A logic program which contains only datalog Horn clauses is called a datalog program. The set of all ground atoms constructed from $\mathcal{P}, \mathcal{C}$ is called the datalog Herbrand Base. $\theta = \{v_1/t_1, \ldots, v_v/t_n\}$ is a substitution in the case that each v_i is a variable and each t_i is a term. $E\theta$ is formed by replacing each variable v_i from θ found in E by t_i. μ is called a unifying substitution for atoms A, B in the case $A\mu = B\mu$. We say clause C θ-subsumes clause D or $C \succeq_\theta D$ whenever there exists a substitution θ such that $C\theta \subseteq D$.

3.2 Framework

We first define the higher-order *meta-rules* used by the Prolog meta-interpreter.

Definition 1[3] (*Meta-rules*) A meta-rule is a higher-order wff

$$\exists\sigma\forall\tau\, P(s_1, \ldots, s_m) \leftarrow \ldots, Q_i(t_1, \ldots, t_n), \ldots$$

where σ, τ are disjoint sets of variables, $P, Q_i \in \sigma \cup \tau \cup \mathcal{P}$ and $s_1, \ldots, s_m, t_1, \ldots, t_n \in \sigma \cup \tau \cup \mathcal{C}$. Meta-rules are denoted concisely without quantifiers as

$$P(s_1, \ldots, s_m) \leftarrow \ldots, Q_i(t_1, \ldots, t_n), \ldots$$

The quantified version of the meta-rules in Fig. 3 is shown in Fig. 4. In general, unification is known to be semi-decidable for higher-order logic (Huet 1975). We now contrast the case for higher-order datalog programs.

Proposition 1 (Decidable unification) *Given higher-order datalog atoms $A = P(s_1, \ldots, s_m)$, $B = Q(t_1, \ldots, t_n)$ the existence of a unifying substitution μ is decidable.*

Proof. *A, B has unifying substitution μ iff $p(P, s_1, \ldots, s_m)\mu = p(Q, t_1, \ldots, t_n)\mu$.*

[3] Note that in the context of meta-rules, only universally quantified variables are not substituted on application within the meta-interpreter, and are thus effectively constants, and so represented in lower case. Existential variables are upper case (as usual).

Meta-Substitution	**Higher-order substitution**
$metasub(instance, [father, ted, bob])$	$\{P/father, X/ted, Y/bob\}$
$metasub(base, [ancestor, p1])$	$\{P/ancestor, Q/p1\}$
$metasub(tailrec, [ancestor, p1, ancestor])$	$\{P/ancestor, Q/p1, R/ancestor\}$

Fig. 5 Relationship between meta-substitutions used by the meta-interpreter and higher-order substitutions

Figure 5 shows the meta-rules from Fig. 3 of the relationship between the meta-substitutions constructed by the meta-interpreter (Fig. 2) and higher-order substitutions of existential variables.

Definition 2 (*MIL setting*) Given meta-rules M, definite program background knowledge B and ground positive and negative unit examples E^+, E^-, MIL returns a higher-order datalog program hypothesis H if one exists such that $M, B, H \models E^+$ and M, B, H, E^- is consistent.

The following describes decidable conditions for MIL.

Theorem 1 (MIL decidable) *The MIL setting is decidable in the case M, B, E^+, E^- are Datalog and $\mathcal{P}, \mathcal{C}$ are finite.*

Proof Follows from the fact that the set of Herbrand interpretations is finite. □

3.3 Learning from interpretations

The MIL setting is an extension of the Normal semantics setting (Muggleton and Raedt 1994) of ILP. We now consider whether a variant of MIL could be formulated within the Learning from Interpretations setting (De Raedt 1997) in which examples are pairs $\langle I, T\rangle$ where I is a set of ground facts (an interpretation) and T is a truth value. In this case each interpretation I will be a subset of the datalog Herbrand Base, which is constructed from $\mathcal{P}$ and $\mathcal{C}$. To account for predicate invention we assume that $\mathcal{P}' \subseteq \mathcal{P}$ and $\mathcal{C}' \subseteq \mathcal{C}$ represent uninterpreted predicates and constants respectively, whose interpretation is assigned by the Meta-interpreter. Since the user has no ascribed meaning for $\mathcal{P}'$ and $\mathcal{C}'$, it would not be possible to provide examples containing full interpretations, and therefore the Learning from Interpretations setting is inappropriate. Clearly, a variant of the Learning from Interpretations setting could be introduced in which examples are represented as incomplete interpretations.

3.4 Language classes and expressivity

We now define language classes for instantiated hypotheses.

Definition 3 (*H^i_j program class*) Assuming i, j are natural, the class H^i_j contains all higher-order definite datalog programs constructed from signatures $\mathcal{P}, \mathcal{C}$ with predicates of arity at most i and at most j atoms in the body of each clause.

The class of dyadic logic programs with one function symbol has Universal Turing Machine (UTM) expressivity (Tärnlund 1977). Note that H^2_2 is sufficient for the kinship example in Sect. 1. This fragment also has UTM expressivity, as demonstrated by the following H^2_2 encoding of a UTM in which $S, S1, T$ represent Turing machine tapes.

$$utm(S, S) \leftarrow halt(S).$$
$$utm(S, T) \leftarrow execute(S, S1), utm(S1, T).$$
$$execute(S, T) \leftarrow instruction(S, F), F(S, T).$$

We assume the UTM has a suitably designed set of machine instructions representing functions of the form

$$f : \mathcal{T} \rightarrow \mathcal{T}$$

where $\mathcal{T}$ is the set of all Turing machine tapes. Below assume G is a datalog goal and program $P \in H^2_2$.

Proposition 2 (Undecidable fragment of H^2_2) *The satisfiability of* G, P *is* undecidable *when* $\mathcal{C}$ *is infinite.*

Proof Follows from undecidability of halting of UTM above. □

The situation differs in the case $\mathcal{C}$ is finite.

Theorem 2 (Decidable fragment of H^2_2) *The satisfiability of* G, P *is* decidable *when* $\mathcal{P}, \mathcal{C}$ *is finite.*

Proof The set of Herbrand interpretations is finite. □

The universality of the H^2_2 fragment is established by the existence of the UTM above. However, there are many concepts for which this approach would lead to a cumbersome and inefficient approach to learning. For instance, consider the following definition of an undirected edge.

$$undirected(A, B) : -edge(A, B), edge(B, A).$$

A program for the UTM above would be required to search through each pair of edges to find a pair of nodes A, B with associated edges in each direction. As a deterministic program this is non-trivial and is likely to involve two iterative loops. However, the clause above is directly and compactly representable within H^2_2 given the following meta-rule.

$$P(x, y) \leftarrow Q(x, y), R(y, x)$$

Thus effective use of meta-rules can lead to a more constrained and effective search. Although the meta-interpreter in Fig. 2 can be applied to meta-rules containing more than two atoms in the body, we limit ourselves to the H^2_2 fragment throughout this paper. This restriction limits the number of possible meta-rules and forces more prolific predicate invention, leading to more opportunities for internal re-use of part of the hypothesised program.

4 Implementation

The Metagol$_D$ system is an implementation of the MIL setting in Yap Prolog and is based around the general Meta-Interpreter shown in Sect. 1. The modifications are aimed at increasing efficiency of a Prolog backtracking search which returns the first satisfying solution of a goal

$$\leftarrow \ldots e^+_i, \ldots, not(e^-_j), \ldots$$

where $e^+_i \in E^+$ and $e^-_j \in E^-$ and *not* represents negation by failure. In particular, the modifications include methods for a) ordering the Herbrand Base, b) returning a minimal cardinality hypothesis, c) logarithmic clause bounding and d) use of a series of episodes for ordering multi-predicate incremental learning.

Lexicographic	**Interval inclusion**
parent(a_alice,b_ted)	leq(0,0)
..	leq(1,1)
parent(c_jake,d_john)	leq(2,2)
..	..
grandparent(a_alice,e_jane)	leq(0,1)
grandparent(c_jake,f_bob)	leq(1,2)
..	leq(0,2)
Lex OrderTest	**Inclusion OrderTest**
$P \succ_{\mathcal{P}} Q$ AND $P \succ_{\mathcal{P}} R$	$x \succ_{\mathcal{C}} z$ AND $z \succ_{\mathcal{C}} y$

Fig. 6 Datalog Herbrand Base orderings with chain meta-rule OrderTests

4.1 Ordering the Herbrand Base

Within ILP, search efficiency depends on the partial order of θ-subsumption (Nienhuys-Cheng and de Wolf 1997). Similarly in Metagol$_D$ search efficiency is achieved using a total ordering over the Herbrand Base to constrain deductive, abductive and inductive operations carried out by the Meta-Interpreter and to reduce redundancy in the search. In particular, we employ (Knuth and Bendix 1970; Zhang et al. 2005) (lexicographic) as well as interval inclusion total orderings over the Herbrand Base to guarantee termination. Termination is guaranteed because atoms higher in the Herbrand Base are always proved by ones lower in the ordering. Also the ordering is finite and so cannot be infinitely descending.

The user input file for Metagol contains an initial list of predicate symbols and constants. The ordering of these lists provide the basis for the total orderings $\succeq_{\mathcal{P}}$ and $\succeq_{\mathcal{C}}$.

Example 1 (Predicate constant symbol ordering) The initial predicate and constant symbol orderings for the kinship example are as follows.

```
initial_predicates([mother/2,father/2]).
initial_constants([matilda,jake,mary,john,bill,
                   alice,andy,susan,harry,liz,
                   megan,ted,jill,jo,sam,jane,bob]).
```

Using the ordering provided by the user in these lists Metagol can infer, for instance, that mother/2 $\succ_{\mathcal{P}}$ father/2 and matilda $\succ_{\mathcal{C}}$ bill.

During an episode (see Sect. 4.4) when a new predicate definition p/a is learned, p/a is added to the head of the list along with a frame of invented auxiliary predicates $p_1, \ldots, p_n$. This allows an ordered scope of predicates which can be used to define p/a consisting of local invented auxiliary predicates, followed by predicates from preceding episodes, followed by any initial predicates.

Figure 6 illustrates alternative OrderTests which each constrain the chain meta-rule to descend through the Herbrand Base. In the *lexicographic* ordering predicates which are higher in the ordering, such as *grandparent/2*, are defined in terms of ones which are lower, such as *parent/2*.

Example 2 (Lexicographic order) The definite clause

$$\text{grandparent(X,Y)} \leftarrow \text{parent(X,Z), parent(Z,Y)}$$

Fig. 7 Recursive definition for *even/1* learned by Metagol using a modified interval inclusion constraint. The invented predicates correspond to *even_1=predecessor* and *even_2=odd*

```
even(0).
even(A) ← even_1(A,B), even_2(B).
even_1(A,B) ← succ(B,A).
even_2(A) ← even_1(A,B), even(B).
```

is consistent with the lexicographic ordering on the chain meta-rule since grandparent/2 $\succ_{\mathcal{P}}$ parent/2.

Meanwhile *interval inclusion* supports definitions of (mutually) recursive definitions such as *leq/2*, *ancestor/2*, *even/2* and *odd/2*.[4]

Example 3 (Interval inclusion order) The definite clause

$$\mathrm{leq(X,Y)} \leftarrow \mathrm{succ(X,Z),\ leq(Z,Y)}$$

is not consistent with a lexicographic ordering on the chain meta-rule since leq $\not\succ_{\mathcal{P}}$ leq. However, it is consistent with the interval inclusion ordering since in the case $X < Z < Y$ the interval $[X, Y]$ includes both $[X, Z]$ and $[Z, Y]$. To ensure interval inclusion holds for the chain rule it is sufficient to test $X \succ_{\mathcal{C}} Z$ and $Z \succ_{\mathcal{C}} Y$. Thus interval inclusion supports the learning of (mutually) recursive predicates. Finite descent guarantees termination even when an ordering is infinitely ascending (e.g. over the natural numbers).

Figure 7 shows a definition of even number learned by Metagol.[5] The solution involves mutual recursion with the definition of odd (invented predicate *even_2*). A modified interval inclusion order constraint forces Metagol to use an inverse meta-rule $P(x, y) \leftarrow Q(y, x)$ to introduce predecessor (invented predicate *even_1*) which then guarantees termination over natural numbers.

4.2 Minimum cardinality hypotheses

Metagol$_D$ uses iterative deepening to ensure the first hypothesis returned contains the minimal number of clauses. The search starts at depth 1. At depth i the search returns an hypothesis consistent with at most i clauses if one exists. Otherwise it continues to depth $i + 1$. During episode p (see Sect. 4.4) at depth i Metagol augments $\mathcal{P}$ with up to $i - 1$ new predicate symbols which are named as extensions of the episode name as $p_1, \ldots p_{i-1}$.

Example 4 (Auxilliary predicate names) In the kinship example, great-grandparent (*ggparent*) can can be learned by Metagol using only the initial predicate definitions for *father/2* and *mother/2*. The minimal logic program found at depth 4 is shown in Fig. 8. In this case predicates are invented which correspond to both *grandparent* and *parent*.

4.3 Logarithmic bounding and PAC model

We now consider the error convergence of a PAC learning model (Valiant 1984) of the logarithmic bounded iterative deepening approach used in Metagol$_D$.

[4] The predicates *even/2* and *odd/2* can be treated as intervals by treating *even(X)* and *odd(Y)* as the natural number intervals $[0, X]$ and $[0, Y]$.

[5] Difficulties involved in learning a mutually recursive definition of even/1 and odd/1 was used by Yamamoto (1997) to demonstrate the incompleteness of Inverse Entailment.

```
ggparent(A,B) ← ggparent_1(A,C), ggparent_2(C,B).
ggparent_1(A,B) ← ggparent_2(A,C), ggparent_2(C,B).
ggparent_2(A,B) ← father(A,B).
ggparent_2(A,B) ← mother(A,B).
```

Fig. 8 Minimal logic program learned by Metagol for great-grandparent (*ggparent*). The invented predicates correspond to *ggparent_1=grandparent* and *ggparent_2=parent*

Lemma 1 (Bound on hypotheses of size d). *Assume m training examples, d is the maximum number of clauses in a hypothesis where $d \leq log_2 m$ and c is the number of distinct clauses in H^2_2 for a given $\mathcal{P}, \mathcal{C}$. The number of hypotheses $|H_d|$ considered at depth d is bounded by $m^{log_2 c}$.*

Proof Since each hypothesis in H_d consists of d clauses chosen from a set of size c it follows that $|H_d| = \binom{c}{d} \leq c^d$. Using the logarithmic bound $c^d \leq c^{log_2 m} = m^{log_2 c}$. Thus $|H_d| \leq m^{log_2 c}$. □

We now consider the size of the space containing all hypotheses up to and including size d.

Proposition 3 (Bound on hypotheses up to size d) *The hypothesis space considered in all depths up to and including d is bounded by $dm^{log_2 c}$.*

Proof Follows from the fact that the number of hypotheses in each depth up to d is bounded by $m^{log_2 c}$. □

We now evaluate the minimal sample convergence for Metagol$_D$.

Theorem 3 *Metagol's logarithmic bounded iterative deepening strategy has a polynomial sample complexity of $m \geq \frac{ln(d)+ln(5)log_2 c+ln\frac{1}{\delta}}{\epsilon}$.*

Proof According to the Blumer bound (Blumer et al. 1989) the error of consistent hypotheses is bounded by ϵ with probability at least $(1-\delta)$ once $m \geq \frac{ln|H|+ln\frac{1}{\delta}}{\epsilon}$, where $|H|$ is the size of the hypothesis space. From Proposition 3 this happens with the Metagol strategy once $m \geq \frac{ln(dm^{log_2 c})+ln\frac{1}{\delta}}{\epsilon}$. Assuming $\epsilon, \delta < \frac{1}{2}$ and $c, d \geq 1$ and simplifying gives $\frac{ln(m)+ln(2)}{m} < \frac{1}{2}$. Numerically this holds once m is at least 5. Substituting into the Blumer bound gives $m \geq \frac{ln(d)+ln(5)log_2 c+ln\frac{1}{\delta}}{\epsilon}$. □

This theorem indicates that in order to ensure polynomial time learning in the worst case we need an example set which is much larger than the definition we aim to learn. We refer to this as the *big data* assumption. In the case that only small numbers of examples are available (see Lin et al. 2014) the big data assumption in Metagol can be over-ridden by giving a maximum bound on the number of clauses (see Sect. 5.3.2).

4.4 Episodes for multi-predicate learning

Learning definitions from a mixture of examples of two inter-dependent predicates such as *parent* and *grandparent* requires more examples and search than learning them sequentially in separate *episodes*. In the latter case the *grandparent* episode is learned once it can use

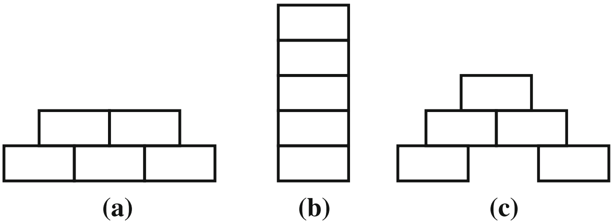


Fig. 9 Examples of **a** stable wall, **b** column and **c** non-stable wall

the definition from the *parent* episode. This phenomenon can be explained by considering that time taken for searching the hypothesis space for the joint definition is a function of the product of the hypothesis spaces of the individual predicates. By contrast the total time taken for sequential learning of episodes is the sum of the times taken for the individual episodes[6] so long as each predicate is learned with low error.

5 Experiments

In this section we describe experiments in which *Metagol* is used to carry out 1) predicate invention for structuring robot strategies and 2) predicate invention and recursion learning for east–west trains and 3) construction of higher-order and first-order concepts for language learning using data from the NELL project (Carlson et al. 2010). All datasets together with the implementation of *Metagol* used in the experiments are available at http://ilp.doc.ic.ac.uk/metagolD_MLJ/.

5.1 Robot strategy learning

In AI, planning traditionally involves deriving a sequence of actions which achieves a specific goal from a specific initial situation (Russell and Norvig 2010). However, various machine learning approaches support the construction of strategies.[7] Such approaches include the SOAR architecture (Laird 2008), reinforcement learning (Sutton and Barto 1998), and action learning within ILP (Moyle and Muggleton 1997; Otero 2005).

In this experiment structured strategies are learned which build a stable wall from a supply of bricks. Predicate invention is used for top-down construction of re-usable sub-strategies. Fluents are treated as monadic predicates which apply to a situation, while Actions are dyadic predicates which transform one situation to another.

5.1.1 Materials

Figure 9 shows a positive example (a) of a stable wall together with two negative examples (unstable walls) consisting of a column (b) and a wall with insufficient central support (c). Predicates are either *high-level* if defined in terms of other predicates or *primitive* otherwise. High-level predicates are learned as datalog definitions. Primitive predicates are non-datalog background knowledge which manipulate situations as compound terms.

[6] Misordering episodes leads to additional predicate invention.

[7] A strategy is a mapping from a set of initial to a set of goal situations.

$$\boxed{\begin{array}{l} buildWall(X,Y) \leftarrow buildWall\_1(X,Y),\ emptyResource(Y) \\ buildWall(X,Y) \leftarrow buildWall\_1(X,Z),\ buildWall(Z,Y) \\ buildWall\_1(X,Y) \leftarrow fetch(X,Z),\ putOnTopOf(Z,Y) \end{array}}$$

Fig. 10 Column/wall building strategy learned from positive examples. *buildWall_1* is invented

$$\boxed{\begin{array}{l} buildWall(X,Y) \leftarrow buildWall\_1(X,Y),\ buildWall\_3(Y) \\ buildWall(X,Y) \leftarrow buildWall\_1(X,Z),\ buildWall(Z,Y) \\ buildWall\_1(X,Y) \leftarrow buildWall\_2(X,Y),\ buildWall\_3(Y) \\ buildWall\_2(X,Y) \leftarrow fetch(X,Z),\ putOnTopOf(Z,Y) \\ \ buildWall\_3(X) \leftarrow offset(X),\ continuous(X) \end{array}}$$

Fig. 11 Stable wall strategy built from positive and negative examples. buildWall_2, buildWall_1 and buildWall_3 are invented

A wall is a list of lists. Thus Fig. 9a can be represented as [[2, 4], [1, 3, 5]], where each number corresponds to the position of a brick[8] and each sublist corresponds to a row of bricks. The primitive actions are *fetch* and *putOnTopOf*, while the primitive fluents are *emptyResource*, *offset* and *continuous* (meaning no gap). This model is a simplification of a real-world robotics application.

When presented with only positive examples, Metagol$_D$ learns the recursive strategy shown in Fig. 10. The invented action $buildWall\_2$ is decomposed into sub-actions *fetch* and *putOnTopOf*. The strategy is non-deterministic and repeatedly fetches a brick and puts it on top of others so that it could produce either Fig. 9a or b.

Given negative examples Metagol$_D$ generates the refined strategy shown in Fig. 11, where the invented action $buildWall\_1$ tests the invented fluent $buildWall\_3$. $buildWall\_3$ can be interpreted as *stable*. This revised strategy will only build stable walls like Fig. 9a.

5.1.2 Method

An experiment was conducted to compare the performance of Metagol$_D$ against that of Progol. Training and test examples of walls containing at most 100 bricks were randomly selected without replacement. Training set sizes were {2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 30, 40, 50, 60, 70, 80} and the test set size was 200. Both training and test datasets contain half positive and half negative, thus the default accuracy is 50 %. Predictive accuracies and associated learning times were averaged over five resamples for each training set size.

5.1.3 Results and discussion

Metagol's accuracy and learning time plots shown in Fig. 12 indicate that, consistent with the analysis in Sect. 4.3, Metagol$_D$, given increasing number of randomly chosen examples, produces rapid error reduction while learning time increases roughly linearly. The dramatic time increase at training size 10 is due to the additional negative example, which requires Metagol$_D$ switching to a different hypothesis with larger size. Figure 13 shows such an example. Originally, Metagol$_D$ derived the hypothesis H_8 for a set of eight training example, but the additional negative example [[9],[6,8,10],[2,6,8,10]], which is explainable by H_8, forces Metagol$_D$ to backtrack and continue the search until H_{10} is found.

[8] Bricks are width 2 and position is a horizontal index.

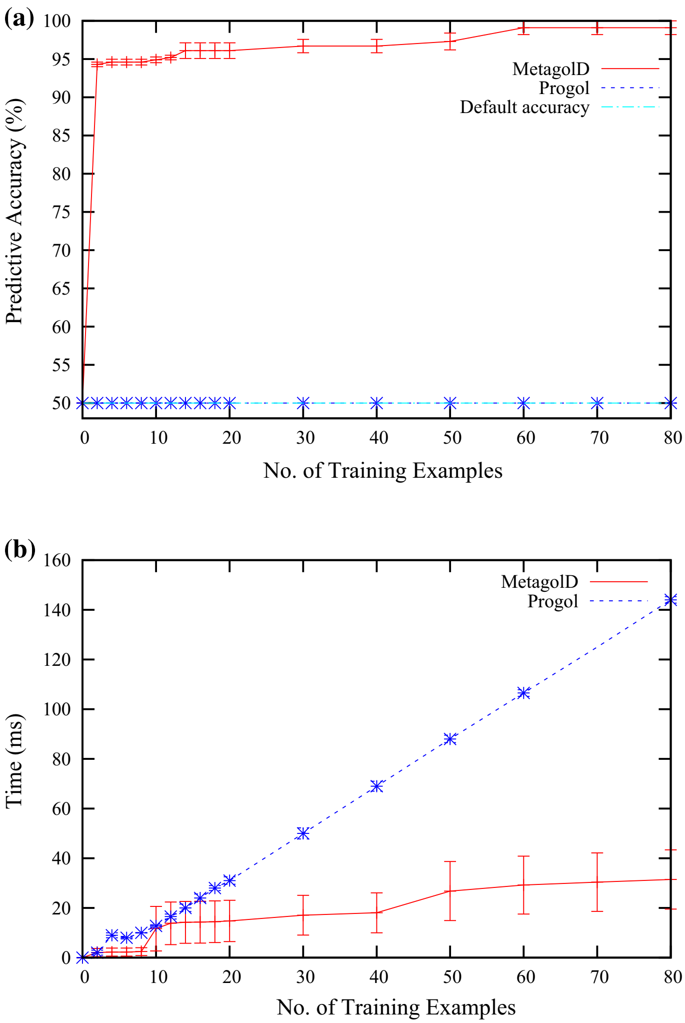


Fig. 12 Graphs of **a** predictive accuracy and **b** learning time for robot strategy learning

H_8	$buildWall(X,Y) \leftarrow buildWall\_1(X,Y),\ offset(Y)$ $buildWall(X,Y) \leftarrow buildWall\_1(X,Z),\ buildWall(Z,Y)$ $buildWall\_1(X,Y) \leftarrow fetch(X,Z),\ putOnTopOf(Z,Y)$
H_{10}	$buildWall(X,Y) \leftarrow buildWall\_1(X,Y),\ emptyResource(Y)$ $buildWall(X,Y) \leftarrow buildWall\_1(X,Z),\ buildWall(Z,Y)$ $buildWall\_1(X,Y) \leftarrow buildWall\_2(X,Y),\ continuous(Y)$ $buildWall\_2(X,Y) \leftarrow fetch(X,Z),\ putOnTopOf(Z,Y)$

Fig. 13 H_8: an hypothesis derived by Metagol$_D$ at training size 8; H_{10}: an hypothesis derived by Metagol$_D$ at training size 10

At training size 2 with only one positive and one negative example, Metagol$_D$ already reaches a predictive accuracy in excess of 90 %. This shows the small sample complexity of Metagol$_D$ due to the inductive bias incorporated in its meta-interpreter.

Figure 12 compares the performance of Progol on this problem. Progol is not able to derive the theory shown in Fig. 11 due to its limitations for learning recursive theories and predicate invention. The only hypothesis derivable by Progol is $buildWall(A, B) \leftarrow fetch(A, C), putOnTop(C, B)$, which only tells how to build a wall out of one single brick. Given that training and test examples are dominated by stable walls with more than one brick, Progol's hypothesis has default accuracy in Fig. 12.

5.2 East–west trains

In this section we demonstrate that the Dyadic representation considered in this paper is sufficient for learning typical ILP problems and show the advantage of *Metagol* when predicate invention and recursion learning is required. In particular we use *Metagol* to discover logic programs for classifying Michalski-style east–west trains from a machine learning competition (Michie 1994; Michie et al. 1994). This competition was based on a classification problem first proposed by Larson and Michalski (1977) which has been regarded as a classical machine learning problem and many real-world problems (e.g. non-determinate learning of chemical properties from atom and bond descriptions) can be mapped to Michalski's trains problem.

5.2.1 Materials

Michalski's original trains problem is shown in Fig. 14. The learning task is to discover a general rule that distinguishes five eastbound trains from five westbound trains. An example of such a rule is, *If a train has a car which is short and closed then it is eastbound and otherwise westbound*. Learning a classifier from the original 10 trains is not difficult for most of existing ILP systems. However, more challenging trains problems have been introduced based on this. For example, Michie et al. (1994; Michie 1994) ran a machine learning challenge competition which extended Michalski's original 10 trains with 10 new trains as shown in Fig. 15. The challenge involved using machine learning systems to discover the simplest possible theory which can classify the combined set of 20 trains of Figs. 14 and 15 into eastbound and westbound trains. The complexity of a theory was measured by the sum of the number of occurrences of clauses, atoms and terms in the theory. A second competition involved classifying 100 separate trains, using a classifier learned from the 20 trains of Figs. 14 and 15.

5.2.2 Method

In this section we use *Metagol* to learn a theory using the 20 trains from the first competition and evaluate the learned theory using the 100 trains from the second competition described above.

5.2.3 Results and discussion

When provided with standard background knowledge for this problem and the 20 trains examples (shown in Figs. 14, 15), *Metagol* learns a recursive theory shown in Fig. 16. This theory correctly classifies all 20 trains and is equivalent to the winning entry for the first competition submitted by Bernhard Pfahringer (Turney 1995). His program used a brute-force search to generate all possible clauses of size 3,4,…and so on in a depth-first iterative deepening manner where every completed clause is checked for completeness and coverage.

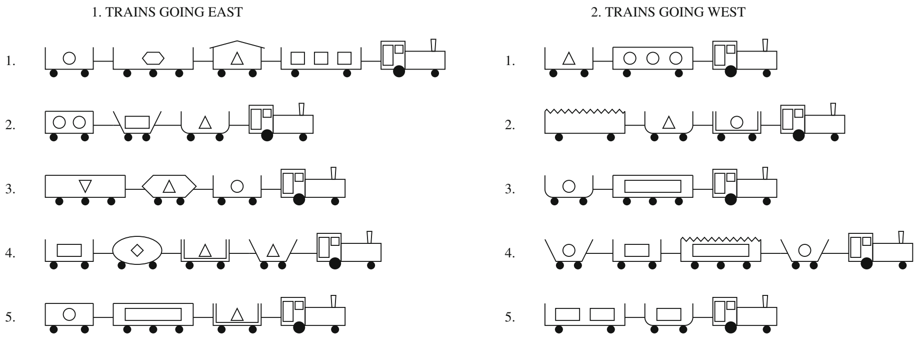


Fig. 14 Michalski's original east–west trains problem

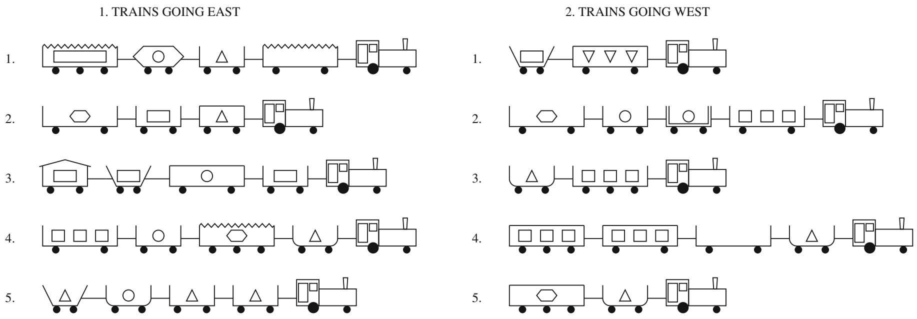


Fig. 15 The new set of 10 trains by Michie et al. (1994) for the east–west competition

$$\boxed{\begin{array}{l} east(A) \leftarrow t1(A), t2(A) \\ t1(A) \leftarrow car(A, B), closed(B) \\ t1(A) \leftarrow cdr(A, B), load1\_triangle(B) \\ t2(A) \leftarrow car(A, B), short(B) \\ t2(A) \leftarrow cdr(A, B), east(B) \end{array}}$$

Fig. 16 A recursive theory found by *Metagol* for the east–west trains competition 1. Predicates $t1$ and $t2$ are invented. *car* and *cdr* provide the first carriage and remaining carriages respectively. *closed*, *short* and $load1\_triangle$ are background knowledge predicates provided in the east–west competitions. $load1\_triangle$ tests if the train has a carriage containing a triangle. This theory correctly classifies all 20 trains and is equivalent to Bernhard Pfahringer's winning entry for the first competition

His program took about 1 day real-time on Sun Sparc 10 (40 MHz). *Metagol* learns the recursive theory of Fig. 16 in about 45 s on a MacBook laptop (2.5 GHz).[9]

The theory found by Pfahringer splits the 100 trains of the second competition into 50 eastbound and 50 westbound trains. We use these trains as the test examples for evaluating hypotheses learned by *Metagol*. The performance of *Metagol* was evaluated on predictive accuracies and running time using the training and test examples described above. The size of training set varied using random samples from the 20 trains (4, 8, 12, 16 and 20) where half were eastbound (positive examples) and half were westbound (negative examples). The predictive accuracies and running time were averaged over 10 randomly sampled training

[9] Ignoring hardware differences other than clockspeed this represents a speed-up of around 30 times over Pfahringer's implementation.

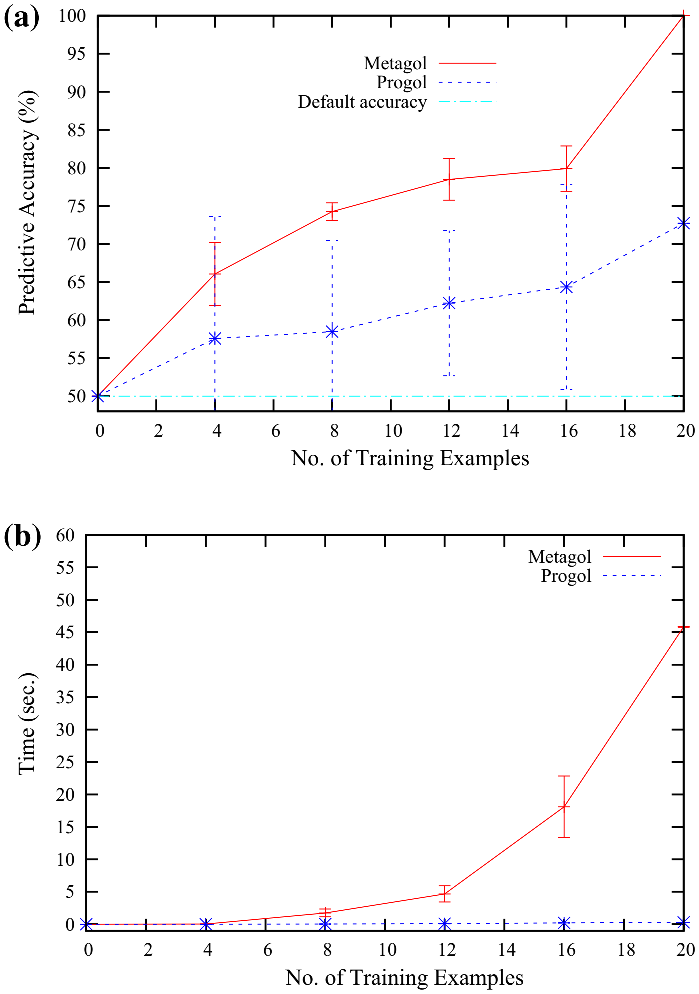


Fig. 17 Predictive accuracies (**a**) and learning times (**b**) of *Metagol* and *Progol* in east–west trains competition 1

examples. For each sample size, we used a fixed test set of 100 trains from the second competition as classified by Pfahringer's model.

The results of the experiments described above are shown in Fig. 17. This figure also compares the performance of *Progol* on this problem. When provided with all 20 trains, *Progol* can quickly find a partial solution shown in Fig. 18. However, due to its limitations for learning recursive theories and predicate invention, *Progol* is not able to find any complete theory for this problem.

5.3 NELL learning

NELL Carlson et al. (2010) is a Carnegie Mellon University (CMU) online system which has extracted more than 50 million facts since 2010 by reading text from web pages. The

$$\boxed{\begin{array}{l} east(A) \leftarrow car(A,B), closed(B), short(B) \\ east(A) \leftarrow cdr(A,B), east(B) \end{array}}$$

Fig. 18 A recursive theory found by *Progol* for the east–west trains competition 1. This is a partial solution and does not cover all examples

facts cover everything from tea drinking to sports personalities. In this paper, we focus on a susbset of NELL about sports, which was suggested by the NELL developers. NELL facts are represented in the form of dyadic ground atoms of the following kind.

playssport(eva_longoria,baseball)
playssport(pudge_rodriguez,baseball)
athletehomestadium(chris_pronger,honda_center)
athletehomestadium(peter_forsberg,wachovia_center)
athletealsoknownas(cleveland_browns,buffalo_bills)
athletealsoknownas(buffalo_bills,cleveland_browns)

5.3.1 Initial experiment: debugging NELL using abduction

A variant of the ILP system FOIL (Quinlan 1990) has previously been used (Lao et al. 2011) to inductively infer clauses similar to the following from the NELL database.

$$athletehomestadium(X, Y) \leftarrow athleteplaysforteam(X, Z), teamhomestadium(Z, Y)$$

In our initial experiment *Metagol* inductively inferred the clause above from NELL data and used it to abduce the following facts, not found in the database.

1. athleteplaysforteam(john_salmons,los_angeles_lakers)
2. athleteplaysforteam(trevor_ariza,los_angeles_lakers)
3. athleteplaysforteam(shareef_abdur_rahim,los_angeles_lakers)
4. athleteplaysforteam(armando_marsans,cincinnati)
5. teamhomestadium(carolina_hurricanes,rbc_center)
6. teamhomestadium(anaheim_angels,angel_stadium_of_anaheim)

Abductive hypotheses 2,4,5 and 6 were confirmed correct using internet search queries. However, 1 and 3 are erroneous. The problem is that NELL's database indicates that only Los Angeles Lakers has Staples Center as its home stadium. In fact Staples is home to four teams.[10] The *Metagol* abductive hypotheses thus uncovered an error in NELL's knowledge[11] which assumed uniqueness of teams associated with a home stadium. This demonstrates MIL's potential for helping debug large scale knowledge structures.

[10] Los Angeles Lakers, Clippers, Kings and Sparks.

[11] Tom Mitchell and Jayant Krishnamurthy (CMU) confirmed these errors and the correctness of the inductively inferred clause.

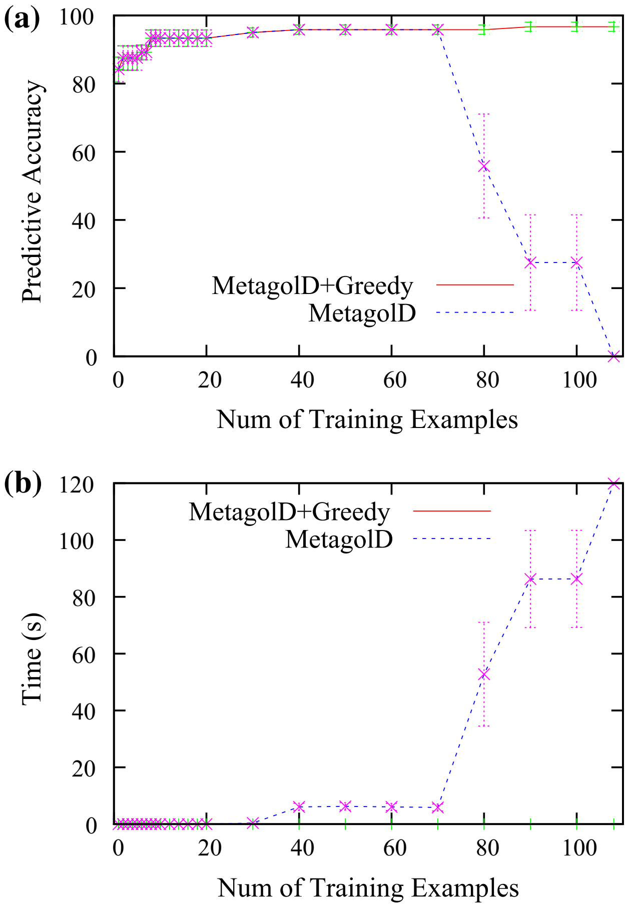


Fig. 19 Predictive accuracies (**a**) and learning times (**b**) of *Metagol* and *Progol* in NELL learning

5.3.2 Evaluation experiment: learning athletehomestadium

We conducted a tenfold cross-validation on this dataset. The training examples were for *athletehomestadium* and consisted of 120 examples in total.[12] They were randomly permuted and divided into 10 folds. During the cross-validation, each fold with 12 examples was used as a test set, while the rest 108 examples were used for training. We considered different training sizes of [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 30, 40, 50, 60, 70, 80, 90, 100, 108]. When the training size was smaller than 108, it was derived from the first two examples of the corresponding set of 108 examples. Figure 19 plots the averaged results on these 10 different folds.

[12] We removed examples which have missing information about *athleteplaysforteam*, considering the specific facts about *athleteplaysforteam* do not have predictive power on examples of *athletehomestadium*. Therefore, the total number of examples decrease from 187 to 120.

The background knowledge contained 919 facts about *athleteplaysforteam* and *teamhomestadium*. However, this data is incomplete. Specifically, 17 % of the examples have incomplete information on *teamhomestadium*, requiring abduction of ground facts. This leads to a large hypothesis space. Below is an example of the target hypothesis, which contains both induced rules and abduced facts.

```
athletehomestadium(A,B) :- athleteplaysforteam(A,C), teamhomestadium(C,B)
teamhomestadium(anaheim_angels,angel_stadium_of_anaheim)
teamhomestadium(blackhawks,united_center)
teamhomestadium(boston_bruins,scotiabank_place)
teamhomestadium(carolina_hurricanes,rbc_center)
teamhomestadium(red_wings,scottrade_center)
teamhomestadium(seattle_mariners,great_american_ballpark)
```

There are seven clauses in this hypothesis. When learning up to seven clauses, not only is the hypothesis space excessively large but learning requires more training examples than is consistent with the logarithmic bound in Theorem 3. Thus in this experiment, we dropped the logarithmic bound condition and instead used a time limit of 2 min. That is, if Metagol$_D$ fails to find a hypothesis within 2 min, then it will return no hypothesis and have predictive accuracy of 0. Consequently the learning curve of Metagol$_D$ (Fig. 19a) drops significantly at training size 80. This is a limitation of the current version Metagol$_D$. However, we investigated addressing this using a greedy search strategy, which is a variant of the dependent learning approach introduced in Lin et al. (2014). Specifically, a shortest hypothesis covering one example is greedily added to the hypothesis so far and becomes part of the background knowledge for learning later examples. The learning curve of 'MetagolD+Greedy' shows the feasibility of such an approach. Considering there are only one or two clauses to be added for each example, the search space is much smaller than the case without the greedy approach. Therefore, the running time with the greedy search strategy is around 0.004 s, which is hundreds of times faster than the original MetagolD without being greedy. However, this greedy search strategy has not been applied to other experiments. The approach seems promising though there are unresolved issues relating to potential overfitting. Further work on these issues is discussed in Sect. 6.1.

NELL presently incorporates manual annotation on concepts being symmetric or transitive. The following meta-rule allows Metagol$_D$ to abduce symmetry of a predicate.

$$P(X, Y) \leftarrow symmetric(P), P(Y, X)$$

Using this Metagol$_D$ abduced the following hypothesis.

$symmetric(athletealsoknownas) \leftarrow$
$athletealsoknownas(buffalo\_bills, broncos) \leftarrow$
$athletealsoknownas(buffalo\_bills, kansas\_city\_chiefs) \leftarrow$
$athletealsoknownas(buffalo\_bills, cleveland\_browns) \leftarrow$

This example demonstrates the potential for the MIL framework to use and infer higher-order concepts.

5.3.3 Predicate invention and recursion

The following hypothesised program was learned by Metagol$_D$ from examples drawn from the NELL database with the facts about $teamplayssport/2$ being removed.[13]

$$\begin{array}{l}
athleteplayssport(X, Y) \leftarrow p1(X, Z), athleteplayssport(Z, Y).\\
p1(X, Y) \leftarrow athleteplaysforteam(X, Z), p2(Z, Y).\\
p2(X, Y) \leftarrow athleteplaysforteam(Y, X).
\end{array}$$

Note $p1$ can be interpreted as $team\_mate$ and $p2$ as the inverse of $athleteplaysforteam$. When inspecting this hypothesis Tom Mitchell and William Cohen commented that PROPPR (Wang et al. 2013) had already learned the following equivalent rule.

$$\begin{aligned}
athleteplayssport(X, Y) \leftarrow & athleteplaysforteam(X, Z),\\
& athleteplaysforteam(W, Z),\\
& athleteplayssport(W, Y).
\end{aligned}$$

This rule can be produced by unfolding the invented predicates in the Metagol$_D$ hypothesis. The advantage of the Metagol$_D$ solution over the PROPPR one is that the invented predicates can be reused as additional background predicates in further learning such as learning another clause for $athletehomestadium/2$, such as $athletehomestadium(X, Y) \leftarrow p1(X, Z), athletehomestadium(Z, Y)$.

5.3.4 Discussion

The NELL experiments are distinct from those on robot strategies and east–west trains in providing an initial indication of the power of the technique to reveal new and unexpected insights in large-scale real-world data. The experiment also indicates the potential for learning higher-order concepts like *symmetric*. Clearly further in-depth work is required in this area to clarify the opportunities for predicate invention and the learning of recursive definitions.

6 Conclusions and further work

MIL (Muggleton et al. 2014) is an approach which uses a Declarative Machine Learning (De Raedt 2012) description in the form of a set of meta-rules, with procedural constraints incorporated within a Meta-Interpreter. The paper extends the theory, implementation and experimental application of MIL from grammar learning to the dyadic datalog fragment H^2_2. This fragment is shown to be Turing expressive in the case of an infinite signature, but decidable otherwise. We show how meta-rules for this fragment can be incorporated into a Prolog Meta-Interpreter. MIL supports hard tasks such as Predicate Invention and learning of recursive definitions by saving successful higher-order substitutions for the meta-rules, which can

[13] In the case where facts about $teamplayssport/2$ are available, then it is sufficient to hypothesis a single clause $athleteplayssport(X, Y) \leftarrow athleteplaysforteam(X, Z), teamplayssport(Z, Y)$.

be used to reconstitute the clauses in the hypothesis. The MIL framework described in this paper has been implemented in Metagol$_D$, which is a Yap Prolog program, and has been made available at http://ilp.doc.ic.ac.uk/metagolD_MLJ/ as part of the experimental materials associated with this paper. However, the approach can also be implemented within more sophisticated solvers such as ASP (see Muggleton et al. 2014). In the Prolog implementation, Metagol$_D$, efficiency of search is achieved by constraining the backtracking search in several ways. For instance, a Knuth–Bendix style total ordering can be imposed over the Herbrand base which requires predicates which are higher in the ordering to be defined in terms of lower ones. Alternatively, an interval inclusion ordering ensures finite termination of (mutual) recursion in the case that the Herbrand Base is infinitely ascending but finitely descending. Additionally, under the *big data* assumption (Sect. 4.3) iterative deepening combined with logarithmic bounding of episodes guarantees polynomial-time searches which identify minimal cardinality solutions. Blumer-bound arguments are provided which indicate that search constrained in this way achieves not only speed improvements but also reduction in out-of-sample error.

We have applied Metagol$_D$ to the problem of inductively inferring robot plans and to NELL language learning tasks. In the planning task the Metagol$_D$ implementation used predicate invention to carry-out top-down construction of strategies for building both columns and stable walls. Experimental results indicate that, as predicted by Theorem 3, when logarithmic bounding is applied, rapid predictive accuracy increase is accompanied by polynomial (near linear) growth in search time with increasing training set sizes.

In the NELL task abduction with respect to inductively inferred rules uncovered a systematic error in the existing NELL data, indicating that Metagol$_D$ shows promise in helping debug large-scale knowledge bases. Metagol$_D$ was also shown to be capable of learning higher-order concepts such as symmetry from NELL data. Additionally predicate invention and recursion were shown to be potentially tractable useful in the NELL context. In an evaluation experiment it was found that learning on the NELL dataset can lead to excessive runtimes, which are improved using a greedy version of the search Metagol mechanism.

6.1 Further work

This paper has not explored the effect of incompleteness of meta-rules on predictive accuracy and robustness of the learning. However, in Cropper and Muggleton (2014) we address this issue by investigating methods for logical minimisation of full enumerations of dyadic meta-rules. This takes advantage of the fact that the MIL framework described in Sect. 3.2 allows meta-rules to be treated as part of the background knowledge, allowing them, in principle, to be revised as part of the learning. In future we aim to further investigate the issue of automatically revising meta-rules. However, as with the approach described in this paper, effective control mechanisms will be key to making the search tractable.

A related issue is that the user is expected to provide the total orderings $\succeq_{\mathcal{P}}$ and $\succeq_{\mathcal{C}}$ over the initial predicate symbols and constants. In future we intend to investigate the degree to which these orderings can be learned.

In the NELL experiment described in Sect. 5.3.2 we found that a greedy modification of the Metagol$_D$ search strategy leads to considerable speed increases. This provides an interesting topic for further work since the use of a non-greedy complete search for each episode leads to search time increasing exponentially in the maximum number of clauses considered [see search bound in Lin et al. (2014)]. However, the greedy approach can lead to overly specific results. For instance, the greedy strategy leads to the following non-minimal, overly specific program when learning *grandparent*

$$grandparent(A, B) \leftarrow father(A, C), father(C, B). \tag{1}$$

$$grandparent(A, B) \leftarrow mother(A, C), mother(C, B). \tag{2}$$

$$grandparent(A, B) \leftarrow mother(A, C), father(C, B). \tag{3}$$

rather than the target theory of

$$grandparent(A, B) \leftarrow parent(A, C), parent(C, B).$$

This problem could conceivably be overcome using a two-stage learning approach in which clauses (1), (2) and (3) are generalised to the target theory clause. This would require that Metagol be extended to allow generalisation over non-ground clauses. Additionally the process should allow for invention of a predicate equivalent to *parent*.

Finally it is worth noting that a Universal Turing Machine can be considered as simply a meta-interpreter incorporated within hardware. In this sense, meta-interpretation is one of, if not the most fundamental concept in Computer Science. Consequently we believe there are fundamental reasons that Meta-Interpretive Learning, which integrates deductive, inductive and abductive reasoning as higher-level operations within a meta-interpreter, will prove to be a flexible and fruitful new paradigm for Artificial Intelligence.

Acknowledgments We thank Tom Mitchell, William Cohen and Jayant Krishnamurthy for helpful discussions and data from the NELL database. We also acknowledge the support of Syngenta in its funding of the University Innovations Centre at Imperial College. The first author would like to thank the Royal Academy of Engineering and Syngenta for funding his present 5 year Research Chair.

References

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

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka Jr., E. R., & Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. In Proceedings of the twenty-fourth conference on artificial intelligence (AAAI 2010).

Cohen, W. (1994). Grammatically biased learning: Learning logic programs using an explicit antecedent description language. *Artificial Intelligence*, *68*, 303–366.

Cropper, A., & Muggleton, S. H. (2014). Logical minimisation of meta-rules within meta-interpretive learning. In Proceedings of the 24th international conference on inductive logic programming. To appear.

Davis, J., & Domingo, P. (2009). Deep transfer via second-order markov logic. In Proceedings of the twenty-sixth international conference on machine learning, pp. 217–224, San Mateo, CA. Morgan Kaufmann.

De Raedt, L. (1997). Logical seetings for concept learning. *Artificial Intelligence*, *95*, 187–201.

De Raedt, L. (2012). Declarative modeling for machine learning and data mining. In Proceedings of the international conference on algorithmic learning theory, p. 12.

De Jong, G. (1981). Generalisations based on explanations. In IJCAI-81, pp. 67–69. Kaufmann.

Driessens, K., & Ramon, J. (2003). Relational instance based regression for relational reinforcement learning. In ICML, pp. 123–130.

Džeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*(1–2), 7–52.

Feng, C., & Muggleton, S. H. (1992). Towards inductive generalisation in higher order logic. In D. Sleeman & P. Edwards (Eds.), *Proceedings of the ninth international workshop on machine learning* (pp. 154–162). San Mateo, CA: Morgan Kaufmann.

Hoare, C. A. R. (1992). Programs are predicates. In Proceedings of the final fifth generation conference, pp. 211–218, Tokyo. Ohmsha.

Hoare, C. A. R., & Jifeng, H. (2001). Unifying theories for logic programming. In C. A. R. Hoare, M. Broy, & R. Steinbruggen (Eds.), *Engineering theories of software construction* (pp. 21–45). Leipzig: IOS Press.

Huet, G. (1975). A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, *1*(1), 27–57.

Inoue, K., Furukawa, K., Kobayashiand, I., & Nabeshima, H. (2010). Discovering rules by meta-level abduction. In L. De Raedt (Ed.), Proceedings of the nineteenth international conference on inductive logic programming (ILP09) (pp. 49–64). Berlin: Springer. LNAI 5989.

Katz, D., Pyuro, Y., & Brock, O. (2008). Learning to manipulate articulated objects in unstructured environments using a grounded relational representation. In *Robotics: Science and systems*. Citeseer.

Kedar-Cabelli, S. T., & McCarty, L. T. (1987). Explanation-based generalization as resolution theorem proving. In P. Langley (Ed.), *Proceedings of the fourth international workshop on machine learning* (pp. 383–389). Los Altos: Morgan Kaufmann.

Knuth, D., & Bendix, P. (1970). Simple word problems in universal algebras. In J. Leech (Ed.), *Computational problems in abstract algebra* (pp. 263–297). Oxford: Pergamon.

Laird, J. E. (2008). Extending the soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, *171*, 224–235.

Lao, N., Mitchell, T., & Cohen, W. W. (2011). Random walk inference and learning in a large scale knowledge base. In Proceedings of the conference on empirical methods in natural language processing (EMNLP), pp. 529–539.

Larson, J., & Michalski, R. S. (1977). Inductive inference of VL decision rules. *ACM SIGART Bulletin*, *63*, 38–44.

Lin, D., Dechter, E., Ellis, K., Tenenbaum, J. B., & Muggleton, S. H. (2014). Bias reformulation for one-shot function induction. In Proceedings of the 23rd European conference on artificial intelligence (ECAI 2014) (pp. 525–530). Amsterdam: IOS Press.

Lloyd, J. W. (2003). *Logic for learning*. Berlin: Springer.

McCarthy, J. (1999). Making robots conscious. In K. Furukawa, D. Michie, & S. H. Muggleton (Eds.), *Machine intelligence 15: Intelligent agents*. Oxford: Oxford University Press.

Michie, D. (1994). On the rails. Computing Magazine. Magzine article text available from http://www.doc.ic.ac.uk/~shm/Papers/computing.pdf

Michie, D., Muggleton, S. H., Page, C. D., Page, D., & Srinivasan, A. (1994). To the international computing community: A new east-west challenge. Distributed email document available from http://www.doc.ic.ac.uk/~shm/Papers/ml-chall.pdf

Miller, D. (1991). A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, *1*(4), 497–536.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, *1*(1), 47–80.

Moyle, S., & Muggleton, S. H. (1997). Learning programs in the event calculus. In N. Lavrač & S. Džeroski (Eds.), *Proceedings of the seventh inductive logic programming workshop (ILP97), LNAI 1297* (pp. 205–212). Berlin: Springer.

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

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. Kaufmann.

Muggleton, S. H., & Lin, D. (2013). Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. In Proceedings of the 23rd international joint conference artificial intelligence (IJCAI 2013), pp. 1551–1557.

Muggleton, S. H., Lin, D., Chen, J., & Tamaddoni-Nezhad, A. (2014). Metabayes: Bayesian meta-interpretative learning using higher-order stochastic refinement. In Proceedings of the 23rd international conference on inductive logic programming, pp. 1–16.

Muggleton, S. H., Lin, D., Pahlavi, N., & Tamaddoni-Nezhad, A. (2014). Meta-interpretive learning: Application to grammatical inference. *Machine Learning*, *94*, 25–49.

Muggleton, S. H., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming*, *19*(20), 629–679.

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.

Nienhuys-Cheng, S-H., & de Wolf, R. (1997). Foundations of inductive logic programming, LNAI 1228. Berlin: Springer.

Otero, R. (2005). Induction of the indirect effects of actions by monotonic methods. In Proceedings of the fifteenth international conference on inductive logic programming (ILP05), volume 3625, pp. 279–294. Berlin: Springer.

Pahlavi, N., & Muggleton, S. H. (2012). Towards efficient higher-order logic learning in a first-order datalog framework. In *Latest advances in inductive logic programming*. London: Imperial College Press.

Pasula, H., Zettlemoyer, L. S., & Kaelbling, L. P. (2004). Learning probabilistic relational planning rules. In ICAPS, pp. 73–82.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*, 239–266.

Rouveirol, C., & Puget, J-F. (1989). A simple and general solution for inverting resolution. In EWSL-89, pp. 201–210, London. Pitman.

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). New Jersey: Pearson.

Sorower, M. S., Doppa, J. R., Orr, W., Tadepalli, P., Dietterich, T. G., & Fern, X. Z. (2011). Inverting grice's maxims to learn rules from natural language extractions. In *Advances in neural information processing systems*, pp. 1053–1061

Srinivasan, A. (2001). *The ALEPH manual*. Oxford: Machine Learning at the Computing Laboratory, Oxford University.

Stahl, I. (1992). Constructive induction in inductive logic programming: An overview. Technical report, Fakultat Informatik, Universitat Stuttgart.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1). Cambridge: Cambridge University Press.

Tärnlund, S.-A. (1977). Horn clause computability. *BIT Numerical Mathematics*, *17*(2), 215–226.

Turney, P. (1995). Low size-complexity inductive logic programming: The east–west challenge considered as a problem in cost-sensitive classification. NRC report cs/0212039, National Research Council of Canada.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*, 1134–1142.

Wang, W. Y., Mazaitis, K., & Cohen, W. W. (2013). Programming with personalized pagerank: A locally groundable first-order probabilistic logic. In Proceedings of the 22Nd ACM international conference on conference on information & #38; knowledge management, CIKM '13 (pp. 2129–2138). New York, NY: ACM.

Yamamoto, A. (1997). Which hypotheses can be found with inverse entailment? In N. Lavrač & S. Džeroski (Eds.), Proceedings of the seventh international workshop on inductive logic programming, LNAI 1297 (pp. 296–308). Berlin: Springer.

Zhang, T., Sipma, H., & Manna, Z. (2005). The decidability of the first-order theory of Knuth–Bendix order. In *Automated deduction-CADE-20* (pp. 738–738). Berlin: Springer.