Meta-Interpretive Learning of Higher-Order Dyadic Datalog: Predicate Invention Revisited [*]

Stephen Muggleton and Dianhuan Lin
Imperial College London
United Kingdom
{s.muggleton,dianhuan.lin08}@imperial.ac.uk

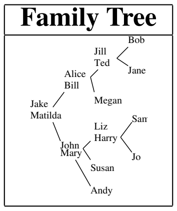
Abstract

In recent years Predicate Invention has been underexplored 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 abduction with respect to a meta-interpreter. New predicate symbols are introduced as constants representing existentially quantified higher-order variables. 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 Dyadic MIL implementation Metagol$_D$ to PAC-learn minimal cardinailty 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 robotic strategies and higher-order concepts in the NELL language learning domain.

1 Introduction

Suppose we machine learn a set of Family Relationships such as those in Figure 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 a hard and largely under-explored topic [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.*, 2013] definite clause grammars are learned using Meta-Interpretive Learning (MIL). In extending MIL to Dyadic Datalog we start by noting each clause of the target theory in Figure 1 takes the form of one of the following *meta-rules*.



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	Meta-form
Examples $ancestor(jake, bob) \leftarrow$ $ancestor(alice, jane) \leftarrow$	$meta1(ancestor, jake, bob) \leftarrow$ $meta1(ancestor, alice, jane) \leftarrow$
Background Knowledge $father(jake, alice) \leftarrow$ $mother(alice, ted) \leftarrow$	$meta1(father, jake, john) \leftarrow$ $meta1(mother, alice, ted) \leftarrow$
Instantiated Hypothesis $father(ted, bob) \leftarrow$ $father(ted, jane) \leftarrow$ $p3(X, Y) \leftarrow father(X, Y)$ $p3(X, Y) \leftarrow mother(X, Y)$ $ancestor(X, Y) \leftarrow p3(X, Y)$ $ancestor(X, Y) \leftarrow p3(X, Z),$ $ancestor(Z, Y)$	**Abduced facts** $meta1(father, ted, bob) \leftarrow$ $meta1(father, ted, jane) \leftarrow$ $meta2(p3, father) \leftarrow$ $meta2(p3, mother) \leftarrow$ $meta2(ancestor, p3) \leftarrow$ $meta3(ancestor, p3, ancestor) \leftarrow$

Figure 1: Family Tree example

Meta-rules
$\mathbf{P}(\mathbf{X}, \mathbf{Y}) \leftarrow$
$\mathbf{P}(\mathbf{x}, \mathbf{y}) \leftarrow \mathbf{Q}(\mathbf{x}, \mathbf{y})$
$\mathbf{P}(\mathbf{x}, \mathbf{y}) \leftarrow \mathbf{Q}(\mathbf{x}, \mathbf{z}), \mathbf{R}(\mathbf{z}, \mathbf{y})$

These three higher-order meta-rules are incorporated (with existential variables **P, Q, R, X, Y** and universal variables **x, y, z**) in the clauses of the following Prolog meta-interpreter.

Meta-Interpreter
$prove(\mathbf{P}, \mathbf{X}, \mathbf{Y}) \leftarrow meta1(P, X, Y).$
$prove(\mathbf{P}, \mathbf{X}, \mathbf{Y}) \leftarrow dyadic(Q), meta2(P, Q), prove(\mathbf{Q}, \mathbf{X}, \mathbf{Y}).$
$prove(\mathbf{P}, \mathbf{X}, \mathbf{Y}) \leftarrow dyadic(Q), dyadic(R), object(Z), meta3(P, Q, R),$ $prove(\mathbf{Q}, \mathbf{X}, \mathbf{Z}), prove(\mathbf{R}, \mathbf{Z}, \mathbf{Y}).$

The two examples shown in the upper part of Figure 1 could be proved using the Prolog goal

$$\leftarrow prove(ancestor, jake, bob), prove(ancestor, alice, jane)$$

[*] The authors would like to acknowledge support from the Royal Academy of Engineering and the Syngenta University Innovation Centre at Imperial College.

given the abduced facts in Figure 1. Note that $p3$ is an invented predicate corresponding to *parent* with the abduced facts corresponding to the *Instantiated Hypothesis*[1] in the lower part of Figure 1.

Completeness of abduction 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 100-1000 times faster than state-of-the-art ILP systems [Muggleton *et al.*, 2013] in the task of learning recursive grammars[2]. In this paper we show how, by suitably constraining the search, MIL's efficiency and completeness advantages can be extended to the broader class of Dyadic Datalog programs. We demonstrated this class to be Turing equivalent, allowing the learning of complex recursive programs such as robot strategies.

The paper is organised as follows. Section 2 describes the MIL framework. The implementation of the Metagol$_D$[3] system is then given in Section 3. Experiments on predicate invention for structuring robot strategies and construction of higher-order concepts for the NELL language learning domain are given in Section 4. In Section 5 we provide a comparison to related work. Lastly we conclude the paper and discuss future work in Section 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 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 $\mathcal{P}$. 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 $\mathcal{C}$. 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 $\{A_1, A_2, ..\neg A_i, \neg A_{i+1}, ...\}$ can be equivalently represented as $(A_1 \vee A_2 \vee ..\neg A_i \vee \neg A_{i+1} \vee ...)$ or $A_1, A_2, .. \leftarrow A_i, A_{i+1}, ...$ 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 $\{C_1, C_2, ...\}$ can be equivalently represented as $(C_1 \wedge C_2 \wedge ...)$. 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. $\theta = \{v_1/t_1, .., v_v/t_n\}$ 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 $C\theta \subseteq D$.

2.2 Framework

We first define the higher-order *meta-rules* incorporated within the Prolog meta-interpreter.

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

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

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

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

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 $A = P(s_1, .., s_m)$, $B = Q(t_1, .., t_n)$ the existence of a unifying substitution μ is decidable.*
Proof. A, B has unifying substitution μ iff $p(P, s_1, .., s_m)\mu = p(Q, t_1, .., t_n)\mu$.

This construction is used to incorporate meta-rules within clauses of the Prolog meta-interpreter shown in Section 1 .

[1] Instantiate meta-rules using *meta1*, *meta2* and *meta3* facts.

[2] Metagol$_R$ and Metagol$_C F$ learn Regular and Context-Free grammars respectively.

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

Definition 2 (Meta-rule incorporation) *The meta-rule $\exists\sigma\forall\tau P(s_1,..,s_m) \leftarrow ..,Q_i(t_1,..,t_n),..$ is incorporated in the Prolog meta-interpreter M iff M contains a clause $Head \leftarrow Body$ where $Head = prove(P,s_1,..,s_m)$ and $Body$ contains $prove(Q_i,t_1,..,t_n)$, $meta_r(\sigma)$ and $type_t(v)$ for each $v \in \tau$.*

The atoms $meta_r(\sigma)$ and $type_t(v)$ provide groundings for all variables within the meta-rule.

Definition 3 (MIL setting) *Given a meta-interpreter 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.*

2.3 Language classes and expressivity

We now define language classes for instantiated hypotheses.

Definition 4 (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 Family example in Section 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) ← halt(S).
utm(S,T) ← execute(S,S1), utm(S1,T).
execute(S,T) ← instruction(S,F), F(S,T).

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 1 (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.

We limit ourselves to the H^2_2 fragment throughout this paper.

3 Implementation

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

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

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-bounding the maximum depth of iterative deepening and d) use of a series of episodes for ordering multi-predicate incremental learning.

Lexicographic	**Interval inclusion**
p1_parent(a_alice,b_ted)	leq(0,0)
..	leq(1,1)
p2_parent(c_jake,d_john)	leq(2,2)
..	..
p3_grandparent(a_alice,e_jane)	leq(0,1)
p3_grandparent(c_jake,f_bob)	leq(1,2)
..	leq(0,2)
Lex OrderTest	**Inclusion OrderTest**
$\langle P,x,y\rangle @ > \langle Q,x,z\rangle$ AND	$x@ > z$ AND
$\langle P,x,y\rangle @ > \langle R,z,y\rangle$	$z@ > y$

Figure 2: Datalog Herbrand Base orderings with chain meta-rule OrderTests. $@ >$ is "lexicographically greater"

3.1 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 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-Bendix [Knuth and Bendix, 1970; Zhang *et al.*, 2005] (lexicograhic) as well as interval inclusion total orderings over the Herbrand Base to guarantee termination. To illustrate, consider the following ordered variant of the chain meta-rule from Section 1.

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

Figure 2 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*, are defined in terms of ones which are lower, such as *parent*. Meanwhile *interval inclusion* supports definitions of (mutually) recursive definitions such as *leq*, *ancestor* and *even/odd*. Interval inclusion assumes all atoms $p(u,v)$ precede the atom $q(x,y)$ in the ordering when the interval $[x,y]$ includes the interval $[u,v]$. Finite descent guarantees termination even when an ordering is infinitely ascending (eg over the natural numbers).

3.2 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 $i+1$.

3.3 Logarithmic bounding and PAC model

In $Metagol_D$ a maximum depth bound for iterative deepening is set as $d = log_2 m$, where m is the number of examples. Assuming c is the number of distinct clauses in H^2_2 for a given $\mathcal{P}, \mathcal{C}$ the number of hypotheses at depth d is

$$|H_d| \leq c^d = 2^{dlog_2 c} = 2^{log_2 m log_2 c} = m^{log_2 c}$$

Since the total number of hypotheses for an iterative deepening strategy is a constant, k, times those at the maximum depth considered, logarithmic bounding ensures the total number of hypotheses considered grows polynomially in m. Within the PAC model [Valiant, 1984] the Blumer bound [Blumer *et al.*, 1989] can be used to show that

$$m \geq \frac{ln(k|H_d|) + ln\frac{1}{\delta}}{\epsilon} = O(ln(m))$$

for the boundary case of $\epsilon = \delta = \frac{1}{2}$. Therefore Metagol$_D$ with logarithmic bounding PAC-learns the class H_2^2.

3.4 Episodes for multi-predicate learning

Learning definitions from a mixture of examples of two interdependent 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 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[4] so long as each predicate is learned with low error.

4 Experiments

In this section we describe experiments in which $Metagol_D$ is used to carry out 1) predicate invention for structuring robot strategies and 2) construction of higher-order concepts for language learning using data from the NELL project [Carlson *et al.*, 2010]. All datasets together with the implementation of $Metagol_D$ used in the experiments are available at http://ilp.doc.ic.ac.uk/metagolD/.

4.1 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[5]. 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 substrategies. Fluents are treated as monadic predicates which apply to a situation, while Actions are dyadic predicates which transform one situation to another.

Figure 3 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.

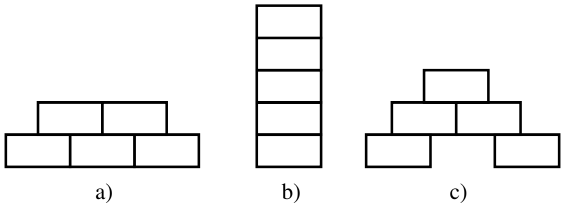


Figure 3: Examples of a) stable wall, b) column and c) non-stable wall

$$\begin{array}{|l|}\hline buildWall(X,Y) \leftarrow a2(X,Y), resourceEmpty(Y) \\ buildWall(X,Y) \leftarrow a2(X,Z), buildWall(Z,Y) \\ a2(X,Y) \leftarrow fetch(X,Z), putOnTopOf(Z,Y) \\ \hline\end{array}$$

Figure 4: Column/wall building strategy learned from positive examples. *a2* is invented.

A wall is a list of lists. Thus Figure 3a) can be represented as $[[2,4],[1,3,5]]$, where each number corresponds to the position of a brick[6] and each sublist corresponds to a row of bricks. The primitive actions are *fetch* and *putOnTopOf*, while the primitive fluents are *resourceEmpty*, *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 Figure 4. The invented action $a2$ is decomposed into subactions *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 Figure 3a or 3b.

Given negative examples Metagol$_D$ generates the refined strategy shown in Figure 5, where the invented action $a2$ tests the invented fluent $f1$. $f1$ can be interpreted as *stable*. This revised strategy will only build stable walls like Figure 3a.

An experiment was conducted to test performance of Metagol$_D$. Training and test examples of walls containing at most 15 bricks were randomly selected with replacement. Training set sizes were $\{2, 4, 8, 16, 32, 64\}$ and the test set size was 1000. 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 10 resamples for each training set size. The accuracy and learning time plots shown in Figure 5 indicate that, consistent with the analysis in Section 3.3, Metagol$_D$, given increasing number of randomly chosen examples, produces rapid error reduction while learning time increases roughly linearly.

4.2 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

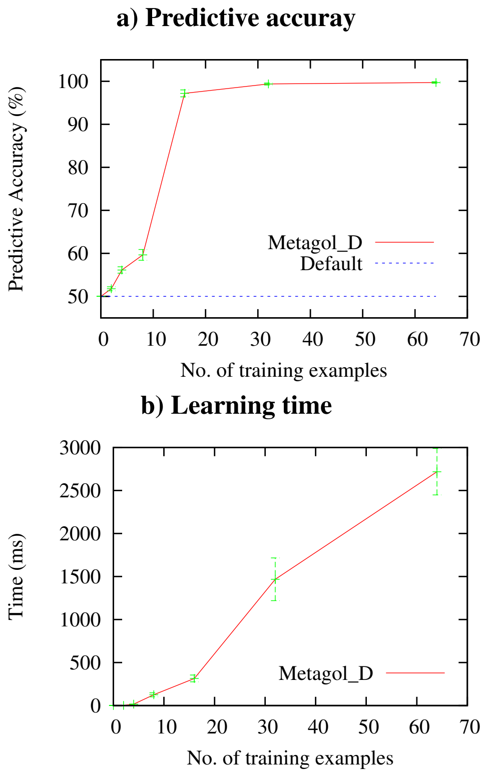
[4] Misordering episodes leads to additional predicate invention.

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

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

$$\begin{array}{l} buildWall(X,Y) \leftarrow a2(X,Y), f1(Y) \\ buildWall(X,Y) \leftarrow a2(X,Z), buildWall(Z,Y) \\ a2(X,Y) \leftarrow a1(X,Y), f1(Y) \\ a1(X,Y) \leftarrow fetch(X,Z), putOnTopOf(Z,Y) \\ f1(X) \leftarrow offset(X), continuous(X) \end{array}$$

Figure 5: Stable wall strategy built from positive and negative examples. a1, a2 and f1 are invented.



a) Predictive accuray

b) Learning time

Figure 6: Graphs of a) Predictive accuracy and b) Learning time for robot strategy learning

facts cover everything from tea drinking to sports personalities and 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)

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 $Metagol_D$ 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[7]. *The $Metagol_D$ abductive hypotheses thus uncovered an error in NELL's knowledge*[8] *which assumed uniqueness of teams associated with a home stadium.* This demonstrates MIL's potential for helping debug large scale knowledge structures.

Learning higher-order concepts

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.

$$\begin{array}{l} symmetric(athletealsoknownas) \leftarrow \\ athletealsoknownas(buffalo\_bills, broncos) \leftarrow \\ athletealsoknownas(buffalo\_bills, kansas\_city\_chiefs) \leftarrow \\ athletealsoknownas(buffalo\_bills, cleveland\_browns) \leftarrow \end{array}$$

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

5 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]), though limited progress has been made in this important topic recently [Muggleton *et al.*, 2011]. In the MIL framework described in [Muggleton *et al.*, 2013] and in this paper, predicate invention is conducted via abduction with respect to a meta-interpreter and by the skolemisation of higher-order variables. This method 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 makes it distinct 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 separately demonstrates MIL's abilities for learning higher-order concepts such as symmetry.

Although John McCarthy long advocated the use of higher-order logic for representing common sense reasoning [McCarthy, 1999], most knowledge representation languages

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

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

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. As a knowledge representation, higher-order datalog (see Section 2), first introduced in [Pahlavi and Muggleton, 2012], has advantages in being both expressive and decidable. The NELL language application demonstrates that higher-order concepts can be readily and naturally expressed in H^2_2 and learned within the MIL framework.

6 Conclusions and further work

MIL [Muggleton *et al.*, 2013] 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-Interpter. MIL supports hard tasks such as Predicate Invention and Recursion via abduction with respect to such a Meta-Interpreter. The MIL framework can be implemented using a simple Prolog program or within a more sophisticated solver such as ASP (see [Muggleton *et al.*, 2013]).

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 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.

Although H^2_2 is theoretically Turing equivalent, some concepts are awkward to express in this fragment. For instance, although symmetry and reflexivity are readily expressed within H^2_2, transitivity requires H^2_3.

In future work we aim to explore the incorporation of probabilities within the MIL framework. In the robot planning domain this would allow modelling of actions and fluents which fail with certain probabilities. Similarly, uncertainty of abductions of player-teams relations could be modified by taking into account the ambiguity of the home stadium relationship in the NELL application. One approach to this might be to incorporate a MIL meta-interpreter into an existing representation such as SLPs [Muggleton, 1996], Blog [Milch and Russell, 2006] or Problog [De Raedt *et al.*, 2007; den Broeck *et al.*, 2010]. If successful these representations could gain from the model estimation methods of MIL, while MIL would gain from the probability estimation approaches within these systems.

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 and Jayant Krishnamurthy for helpful discussions and data from the NELL database.

References

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

[Carlson *et al.*, 2010] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R. Hruschka Jr., and T.M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI 2010)*, 2010.

[De Raedt *et al.*, 2007] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its applications in link discovery. In R. Lopez de Mantaras and M.M Veloso, editors, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 804–809, 2007.

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

[den Broeck *et al.*, 2010] G. Van den Broeck, I. Thon, and M. van Otterlo. DTProbLog: A decision-theoretic probabilistic Prolog. In *Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence, AAAI10*, pages 1217–1222, 2010.

[Feng and Muggleton, 1992] C. Feng and S.H. Muggleton. Towards inductive generalisation in higher order logic. In D. Sleeman and P. Edwards, editors, *Proceedings of the Ninth International Workshop on Machine Learning*, pages 154–162, San Mateo, CA, 1992. Morgan Kaufmann.

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

[Inoue *et al.*, 2010] K. Inoue, K. Furukawa, and I. Kobayashiand H. Nabeshima. Discovering rules by meta-level abduction. In L. De Raedt, editor, *Proceedings of the Nineteenth International Conference on*

Inductive Logic Programming (ILP09), pages 49–64, Berlin, 2010. Springer-Verlag. LNAI 5989.

[Knuth and Bendix, 1970] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon, Oxford, 1970.

[Laird, 2008] J. E. Laird. Extending the soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, pages 224–235, 2008.

[Lao *et al.*, 2011] N. Lao, T. Mitchell, and W.W. Cohen. Random walk inference and learning in a large scale knowledge base. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 529–539, 2011.

[Lloyd, 2003] J.W. Lloyd. *Logic for Learning*. Springer, Berlin, 2003.

[McCarthy, 1999] J. McCarthy. Making robots conscious. In K. Furukawa, D. Michie, and S.H. Muggleton, editors, *Machine Intelligence 15: intelligent agents*. Oxford University Press, Oxford, 1999.

[Milch and Russell, 2006] B. Milch and S. Russell. First-order probabilistic languages: into the unknown. In Stephen Muggleton Ramon Otero and Alireza Tamaddoni-Nezhad, editors, *Proceedings of the 16th International Conference on Inductive Logic Programming*, volume 4455, pages 10–24, 2006.

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

[Moyle and Muggleton, 1997] S. Moyle and S.H. Muggleton. Learning programs in the event calculus. In N. Lavrač and S. Džeroski, editors, *Proceedings of the Seventh Inductive Logic Programming Workshop (ILP97)*, LNAI 1297, pages 205–212, Berlin, 1997. Springer-Verlag.

[Muggleton and Buntine, 1988] S.H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the 5th International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

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

[Muggleton *et al.*, 2013] S.H. Muggleton, D. Lin, D. Pahlavi, and A. Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. In *Proceedings of the 22nd International Conference on Inductive Logic Programming*, 2013. Also invited to associated Machine Learning Journal Special Issue on ILP.

[Muggleton, 1996] S.H. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.

[Nienhuys-Cheng and de Wolf, 1997] S-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag, Berlin, 1997. LNAI 1228.

[Otero, 2005] R. Otero. Induction of the indirect effects of actions by monotonic methods. In *Proceedings of the Fifteenth International Conference on Inductive Logic Programming (ILP05)*, volume 3625, pages 279–294. Springer, 2005.

[Pahlavi and Muggleton, 2012] N. Pahlavi and S.H. Muggleton. Towards efficient higher-order logic learning in a first-order datalog framework. In *Latest Advances in Inductive Logic Programming*. Imperial College Press, 2012. In Press.

[Quinlan, 1990] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Rouveirol and Puget, 1989] C. Rouveirol and J-F. Puget. A simple and general solution for inverting resolution. In *EWSL-89*, pages 201–210, London, 1989. Pitman.

[Russell and Norvig, 2010] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, New Jersey, 2010. Third Edition.

[Srinivasan, 2001] A. Srinivasan. The ALEPH manual. *Machine Learning at the Computing Laboratory, Oxford University*, 2001.

[Stahl, 1992] I. Stahl. Constructive induction in inductive logic programming: an overview. Technical report, Fakultat Informatik, Universitat Stuttgart, 1992.

[Sutton and Barto, 1998] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

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

[Valiant, 1984] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[Zhang *et al.*, 2005] T. Zhang, H. Sipma, and Z. Manna. The decidability of the first-order theory of Knuth-Bendix order. In *Automated Deduction–CADE-20*, pages 738–738. Springer, 2005.