Complete Bottom-Up Predicate Invention in Meta-Interpretive Learning

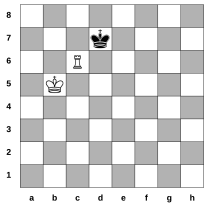
Céline Hocquette , Stephen H. Muggleton
Department of Computing, Imperial College London, London, UK
{celine.hocquette16, s.muggleton}@imperial.ac.uk

Abstract

Predicate Invention in Meta-Interpretive Learning (MIL) is generally based on a top-down approach, and the search for a consistent hypothesis is carried out starting from the positive examples as goals. We consider augmenting top-down MIL systems with a bottom-up step during which the background knowledge is generalised with an extension of the immediate consequence operator for second-order logic programs. This new method provides a way to perform extensive predicate invention useful for feature discovery. We demonstrate this method is complete with respect to a fragment of dyadic datalog. We theoretically prove this method reduces the number of clauses to be learned for the top-down learner, which in turn can reduce the sample complexity. We formalise an equivalence relation for predicates which is used to eliminate redundant predicates. Our experimental results suggest pairing the state-of-the-art MIL system Metagol with an initial bottom-up step can significantly improve learning performance.

1 Introduction

An optimal strategy in the chess endgame KRK (King-and-Rook vs King) is to successively restrict the area available to the opponent's black king using the white rook. Maintenance of this rook safety is achieved using the white king. An example of a situation in which the white king protects its rook is represented in Figure 1a. Figure 1b shows a hypothesis describing such a situation. This hypothesis includes various invented predicates representing sub-concepts. For instance, `f1/2` defines the existence of a white king in a board.
We introduce a new method for partially delegating the construction of invented predicates and demonstrate it can improve learning performance. The hypothesis is divided between surface and substrate predicates as in Figure 1b. The substrate is a set of invented predicates generated in a first step by a bottom-up learner and from the background knowledge. The surface is a hypothesis built subsequently by a top-down learner which can reuse these substrate predicates. The surface hypothesis has fewer clauses owing to the use of substrate predicates and so is easier to learn.



(a) Example board

Surface	`f(S):-f1(S,P1),f3(S,P1).`

Substrate	`f1(S,P1):-f2(S,P1),white(P1).`
	`f2(S,P1):-piece(S,P1),king(P1).`
	`f3(S,P1):-f4(S,P2),distance1(P2,P1).`
	`f4(S,P2):-piece(S,P2),rook(P2).`

(b) Target Hypothesis: S denotes a board state, P1 and P2 are pieces

Figure 1: Learning a chess pattern: the white king protects its rook

In MIL [Muggleton and Lin, 2013], predicate invention is conducted in a top-down fashion by allowing new predicate symbols in metasubstitutions. We introduce a new bottom-up method for performing predicate invention based upon an extension of the immediate consequence operator for second-order logic programs. For each metarule, if body literals can be resolved with the current background knowledge, then Skolem constants are bound to second-order variables in the head. The resulting head is added to the background knowledge and the resulting metasubstitution is saved as a new predicate definition. This process is iterated. We demonstrate our bottom-up method is complete with respect to a fragment of dyadic datalog. Performing bottom-up iterations reduces the number of surface clauses to be learned by the top-down learner which in turn can reduce the sample complexity.
Our contributions are 1) the introduction of a new method for performing extensive predicate invention, 2) a proof of the completeness of this method, 3) the formalisation of the definition of an equivalence relation for predicates which is used to prune redundant predicates, 4) an implementation of this method 5) experimental results over two domains demonstrating this method can significantly improve learning performance.

2 Related Work

2.1 Learning Game Strategies

Early approaches to learning game strategies [Shapiro and Niblett, 1982; Quinlan, 1983] use the decision tree ID3 to classify minimax depth-of-win for positions in chess end game. These approaches use a carefully selected set of board attributes as features. Conversely, we study the automated invention of features and relations from low-level primitives.

2.2 Predicate Invention

Inductive Logic Programming (ILP) systems rely on a language bias to restrict the hypothesis space. Predicate invention is a key challenge in ILP [Muggleton *et al.*, 2012]. It has been investigated as a bias shift [Stahl, 1995] for overcoming the limitations of insufficient vocabulary for learning. New predicates might also allow for the formulation of simpler hypotheses within the target language thus reducing the learning complexity [Stahl, 1993] which we study here for MIL.

Early predicate invention approaches were based on the use of W operators within the inverting resolution framework [Muggleton and Buntine, 1988]. However, the completeness of this approach was never demonstrated. Predicate invention can be performed by adding new predicate symbols to mode declarations [Corapi *et al.*, 2011; Law *et al.*, 2014], however their number and arity has to be user-provided. MIL systems [Muggleton and Lin, 2013] achieve predicate invention in a top-down fashion from the metarules, by introducing Skolem constants representing new predicate symbols. Their number depends on the depth in the iterative deepening search. Conversely, we investigate bottom-up predicate invention in MIL and do not bound the number of invented predicates as such but only bound the depth of the generation process.

Predicate Invention can be performed as a form of meta-learning over time. Dependent Learning [Lin *et al.*, 2014] allows the construction of a series of predicates with increasing levels of abstraction by solving a series of tasks with different complexity. In [Cropper, 2019], predicate invention is performed in an initial unsupervised stage. In both cases, learned hypotheses are saved to the background knowledge as predicate definitions that can be reused when solving subsequent tasks. Both approaches are based on a set of tasks, user-provided or randomly sampled. Conversely, our method builds predicate definitions from the background knowledge related to the examples and does not require additional training tasks.

2.3 Combining Top-Down and Bottom-up

Bidirectional hypothesis search strategy was originally presented in the version space algorithm [Mitchell, 1982]. Variants were implemented in algorithms alternating generalisation and specialisation steps to search through the lattice of clauses [Fensel and Wiese, 1993; Zelle *et al.*, 1994]. Progol [Muggleton, 1995] constructs a most specific clause, the bottom clause, that entails a positive example. In a top-down step, it employs a variant of the A* search to find the best possible consistent definition in the space constrained by the bottom clause. However, [Muggleton, 1995; Fensel and Wiese, 1993] do not support automated predicate invention which restrict their expressivity. [Zelle *et al.*, 1994] includes a mechanism for demand driven predicate invention. Conversely, our system is based upon MIL and fully supports automated predicate invention.

3 Learning Framework

3.1 Logical Notation

We assume familiarity with standard logic programming notations [Lloyd, 1984]. We consider Datalog programs which are definite logic programs without proper function symbols. A variable is second-order if it can be bound to predicate symbols. The Herbrand base $\mathcal{B}_\mathcal{P}$ of a first-order logic program P is the set of all ground atoms constructed with the predicates and constants in P. The process of replacing existential variables by constants in a formula is called Skolemisation. The unique constants are called Skolem constants.

3.2 Meta-Interpretive Learning

MIL [Muggleton and Lin, 2013; Muggleton *et al.*, 2014] is a form of ILP [Muggleton, 1991]. The MIL input is a pair (B, E), where E is a set of ground atoms representing positive and negative examples $E = E^+ \cup E^-$ and B is a second-order logic program $B = B_c \cup M$ composed of a definite first-order background knowledge B_c and metarules M. Metarules are second-order clauses with existentially quantified predicate variables and universally quantified first-order variables (see examples on Figure 2). The MIL problem is to find a hypothesis H such that $B, H \models E$. The proof was originally based upon an adapted Prolog meta-interpreter. It attempts to prove the examples and saves the resulting meta-substitutions for any successful proof. Metasubstitutions are the substitution of second-order variables by predicate symbols. Saved metasubstitutions can be used as background knowledge by substituting them back onto their corresponding metarules. MIL supports predicate invention, the learning of recursions and higher-order programs [Cropper and Muggleton, 2016a]. In the following, we use the MIL system *Metagol* [Cropper and Muggleton, 2016b].

3.3 Immediate Consequence Operator for Predicate Invention

We first recall the well-known definition of the immediate consequence operator [Van Emden and Kowalski, 1976]:

Definition 3.1 (Immediate Consequence Operator of a Definite First Order Program)**.** *Let P be a definite first-order logic program. The Immediate Consequence Operator T_P associated with P is a mapping from subsets of the Herbrand base $\mathcal{B}_\mathcal{P}$ to subsets of $\mathcal{B}_\mathcal{P}$ defined as:*

$$\forall I \subseteq \mathcal{B}_\mathcal{P}, T_P(I) = \{\alpha \in \mathcal{B}_\mathcal{P} \mid \alpha \leftarrow B_1, ..., B_m, m \geq 0 \textit{ is a ground instance of a clause in P and} \{B_1, ..., B_m\} \subseteq I\}$$

MIL systems use second-order logic programs, therefore we extend Definition 3.1 to allow for second-order programs:

Definition 3.2 (Immediate Consequence Operator of a Definite Second Order Program)**.** *Let P be a definite second-order logic program. The Immediate Consequence Operator T_P associated with P is an operator defined over subsets of $\mathcal{B}_\mathcal{P}$ as:*

$$\forall I \subseteq \mathcal{B}_\mathcal{P}, T_P(I) = \{\alpha \mid \alpha \leftarrow B_1, ..., B_m, m \geq 0 \textit{ is a ground instance of a clause in P and } \{B_1, ..., B_m\} \subseteq I \textit{ and second order variables in } \alpha \textit{ are bound to Skolem constants}\}$$

Applying the T_P operator to first order logic programs generates atoms which are part of the Herbrand base of P. For second order logic programs, it generates new atoms which may have Skolem constants as predicate symbols in which case it extends the Herbrand base of P.

Example 3.1. *Suppose there is a chess board with a rook $r1$ and a white king $k1$ located on the same file. We consider a second-order logic program P containing the metarule postcon and ground unit clauses. Given $I = \emptyset$, we have:*

$$P = \{P(X,Y) \leftarrow Q(X,Y), R(Y); rook(r1) \leftarrow; king(k1) \leftarrow; white(k1) \leftarrow; samefile(r1,k1) \leftarrow\}$$
$$T_P(I) = \{rook(r1), king(k1), white(k1), samefile(r1,k1)\}$$
$$T_P(T_P(I)) = \{samefileking(r1,k1), samefilewhite(r1,k1)\}$$

Atoms in $T_P(T_P(I))$ are generated from the following ground instances of the postcon metarule, for which body literals are elements of $T_P(I)$. 'samefileking' and 'samefilewhite' are Skolem constants.

$$samefileking(r1,k1) \leftarrow samefile(r1,k1), king(k1).$$
$$samefilewhite(r1,k1) \leftarrow samefile(r1,k1), white(k1).$$

3.4 Predicate Invention from the T_B Operator

We refer in the following to T_B as the immediate consequence operator associated with the second-order program $B = B_c \cup M$ provided as input to a MIL learner. Predicates are invented as follows. The T_B operator is iteratively applied to the empty set. For each new fact generated, Skolem constants are saved together with the metasubstitutions. These metasubstitutions can later be projected onto the corresponding metarules to derive the definitions of invented predicates.

Example 3.1 (Continued)**.** *Saved metasubstitutions can be projected onto the postcon metarule to derive the following first-order logic program P':*

$$P' = \{samefileking(X,Y) \leftarrow samefile(X,Y), king(Y); samefilewhite(X,Y) \leftarrow samefile(X,Y), white(Y)\}$$

3.5 Elimination of Redundant Predicates

Successive applications of the T_B operator generate a series of predicate symbols together with their definitions. The number of predicate symbols thus monotonically increases with the number of iterations. To avoid cluttering the background knowledge, redundant predicates are pruned at the end of each iteration. We define a notion of equivalence of predicate based upon an equivalence of logic programs [Maher, 1988]. We recall that the success set $SS(P)$ of a first-order logic program P is the set of atoms from the Herbrand base of P which have a successful SLD-derivation for P.

Definition 3.3 (Success set of a predicate)**.** *Given a first-order program P, the success set $SS(p,P)$ of a predicate p is the subset of the success set of P restricted to atoms of p:*

$$SS(p,P) = \{\alpha \in SS(P) | \alpha \textit{ has predicate symbol } p\}$$

Definition 3.4 (Equivalence of Predicates)**.** *Two predicates p_1 and p_2 are equivalent given a first-order program P if they have the same success set up to renaming of the predicate symbols p_1, p_2:*

$$SS(p_1,P) =_{rename(p_1,p_2)} SS(p_2,P)$$

One can verify reflexivity, symmetry and transitivity for the relation defined in Definition 3.4 and conclude it is an equivalence relation. For all predicates p_1, if there exists a predicate p_2 such that p_1 and p_2 are equivalent, p_1 is said to be redundant with respect to p_2. Intuitively, predicates covering exactly the same set of ground atoms are not discriminative in a learning process, and adding redundant predicates in the background knowledge is helpless to build a hypothesis.

The success set of P is equivalent to the least fix point of the immediate consequence operator [Lloyd, 1984]: $SS(P) = T_P \uparrow^\omega$. This provides a practical way of computing success sets, as the new atoms generated at each iteration are saved together with predicate definitions. Therefore, evaluating success sets is straightforward in this context.

Example 3.1 (Continued)**.** *Invented predicates have the success sets below and are equivalent given $P' \cup P$:*

$$SS(samefileking, P' \cup P) = \{samefileking(r1,k1)\}$$
$$SS(samefilewhite, P' \cup P) = \{samefilewhite(r1,k1)\}$$

Algorithm 1 Bottom-Up Learner

Input: second-order logic program $B_{|E}$ related to training examples E, number of iterations k, definitions of initial predicate H_0
Output: logic program H

```
set H = H_0 and I = ∅
for i in [1,k] do
    for all new predicates p from T_B(I) do
        if ∄ p1 ∈ H such that p1 and p are equivalent then
            add the definition of p to H
            add atoms from T_B(I) with predicate symbol p to I
        end if
    end for
end for
```

3.6 Algorithm

Our algorithm for bottom-up predicate construction is presented in Algorithm 1. Given a second-order logic program $B = B_c \cup M$, the learner considers $B_{|E}$, the restriction of B related to the training examples E, and successively computes the immediate consequence of $B_{|E}$ according to Definition 3.2. Success sets of resulting invented predicates are evaluated. Predicates which are not equivalent to any current predicates following Definition 3.4 are saved into the background knowledge. Their success sets are saved such that can be used in subsequent iterations. After k iterations, the top-down learner learns a consistent hypothesis while being allowed to reuse predicates invented in bottom-up iterations.

4 Theoretical Analysis

4.1 Number of Predicate Symbols Introduced

Assumptions 1: We consider the program class H^2_2 which consists of definite Datalog programs with dyadic predicates (arity at most 2) and at most 2 atoms in the body of each clause [Muggleton and Lin, 2013]. We assume the learner is given m metarules in H^2_2 and p initial predicate symbols.

Theorem 4.1 (Number of predicate symbols introduced)**.** *We call y the column vector of powers of p: $y = \{p^j\}_{j=0}^{\infty}$, and e the row vector $e = \{\delta_{j,1}\}_{j=0}^{\infty}$, where $\delta_{j,k}$ is the Kronecker delta. For all $k \in \mathbb{N}^*$, the number of predicate symbols available at the iteration k is upper bounded by a function u_k which is polynomial in p and m and defined by: $\forall\, k \in \mathbb{N}, u_k = eT^k y$, T verifying $\forall\, j,l \in \mathbb{N} : T_{j,l} = \binom{j}{l-j} m^{l-j}$.*

Proof. By induction over $k \in \mathbb{N}$.
$k = 0$: The initial number of predicate symbols p is upper bounded by $u_0 = p$ which is polynomial in p and m.
Let $k \in \mathbb{N}$. Suppose the number of predicate symbols available at the iteration k is bounded by u_k which is polynomial in p and m. The number of different bodies that can be constructed from u_k predicate symbols and a H_2^2 metarule is at most u_k^2. The number of different bodies that can be constructed from m distinct H_2^2 metarules is at most mu_k^2. The number of predicate symbols available at the iteration $k+1$ is bounded by $u_{k+1} = u_k + m.u_k^2$ which is polynomial in p and m. Then, for all $k \in \mathbb{N}$, the number of predicate symbols is bounded by u_k, which is a non linear sequence defined by the recursive formula: $u_0 = p$ and $u_{k+1} = u_k + m.u_k^2$.
The solution to this nonlinear recursive sequence is, from [Rabinovich *et al.*, 1996], the expression given above. □

For a fixed number of iterations k, Theorem 4.1 provides an upper bound on the number of predicate symbols introduced polynomial in p and m. This number of invented predicates is in practice limited by the initial background knowledge B. We recall that, given m metarules in H_2^2 and p initial predicate symbols, top-down MIL complexity for a fixed clause bound n is $\mathcal{O}((mp^3)^n)$ [Lin *et al.*, 2014], which is also a polynomial function of p and m. Our method thus has a worst-case complexity no worst than top-down MIL.

4.2 Completeness

Assumption 2: We assume metarules in M are non-recursive.
We define the following iterated T_B operator as $\forall I \subseteq \mathcal{B}_\mathcal{P}$:

$$T_{0,B}(I) = I$$
$$\forall i \in \mathbb{N}^*, T_{i,B}(I) = T_B(T_{i-1,B}(I)) \cup T_{i-1,B}(I)$$

Algorithm 1 derives a program H containing predicate definitions, each having at most k non-recursive clauses from the hypothesis space. For all clauses in H, there exists a ground substitution θ of the head belonging to $T_{k,B}(\emptyset)$. In other words, H satisfies equation below for k:

$$\forall Head \leftarrow Body \in H, \exists \theta : \begin{cases} Head\,\theta = A \\ A \in T_{k,B}(\emptyset) \end{cases} \quad (1)$$

Theorem 4.2 (Completeness)**.** *Given $k \in \mathbb{N}$ and a theory H within the hypothesis space defined by the primitives and metarules provided, H satisties equation 1 with parameter k only if it is derivable in k iterations of Algorithm 1.*

Proof. By induction over $k \in \mathbb{N}$. For $k = 0$, $H = \emptyset$.
Let $k \in \mathbb{N}$. Assume theories satisfying equation 1 with parameter k are derivable in k iterations of Algorithm 1. We consider a theory H satisfying equation 1 with parameter $k+1$. For all $Head \leftarrow Body \in H$, there exists a ground substitution θ such that $Head\,\theta = A$ and $A \in T_{k+1,B}(\emptyset)$. All literals from $Body\,\theta$ are elements of $T_{k,B}(\emptyset)$. Then all literals from $Body\,\theta$ satisfy Equation (1) thus are derivable in k iterations of Algorithm 1. Performing one more iteration of Algorithm 1 derives $Head\theta \in T_{k+1,B}(\emptyset)$ and the clause $Head \leftarrow Body$ is saved in H. Then, clauses from H are derivable in $k+1$ iterations of Algorithm 1. □

If the top-down learner chosen also is complete, the system combining the bottom-up and top-down learner is complete.

4.3 Sample Complexity

Proposition 4.1 (Sample Complexity gain [Cropper, 2019])**.** *We assume the target hypothesis expressed in its minimal form has n clauses with standard MIL. Let $n - l_k$ be the minimal number of clauses required to express the target theory after k bottom-up iterations. We call p_k the actual number of predicate symbols available after k bottom-up iterations. Given p initial predicate symbols, m metarules in H_2^2, an error level ϵ and a confidence level δ, the number of examples required to achieve an error at most ϵ with confidence δ is reduced after k bottom-up iterations when:*

$$n \ln(p) > (n - l_k) \ln(p_k)$$

By completeness from Theorem 4.2, it is guaranteed that the number of clauses required to express a target theory is reduced by at least $l_k = k$ after k bottom-up iterations.

5 Implementation

Sampling of background knowledge facts. In order to be relevant to the learning task, $B_{|E}$ is the restriction of the background knowledge B related to the examples E. $B_{c|E}$ is a set of ground unit clauses sampled from the examples and initial predicates. First, the ground terms of each examples are extracted. Next, input first-order variables in the head of clauses from the initial background knowledge are instantiated to these ground terms. Every successful resolution of these instantiated heads is saved as a background fact. This process is iterated. If a resolution generates an output ground term, this ground term is saved and can be reused in further resolutions to build new facts. Examples can be sampled to ensure the number of initial facts is not too large.

Applying the T_B operator. New facts and invented predicate definitions are generated by applying the T_B operator from Definition 3.2 to B. Bodies of metarules in M are resolved against the background knowledge. In order to limit redundancy of the proofs, we ensure that each new proof reuses at least one fact proved in the last iteration. Hence a proof executed at the bottom-up iteration i will not be executed again at iteration j, with $j > i$.

Generation of Skolem constants. Skolem constants are built by concatenating the metarule's name and the instantiated second-order variables in the metarule's body. This process ensures the uniqueness of Skolem constants. If two predicates are equivalent with respect to Definition 3.4, the predicate with the shortest number of concatenated names is saved, since it has a simpler definition by construction.

6 Experiments

6.1 Research Hypotheses

We experimentally test within this section whether the use of bottom-up iterations can improve learning performance. Therefore, we investigate the following null hypotheses:

Null Hypothesis 1: *Augmenting MIL systems with bottom-up iterations can not improve predictive accuracies.*
Null Hypothesis 2: *Augmenting MIL systems with bottom-up iterations can not reduce the sample complexity.*

Experiment	Name	Metarule
1	*postcon*	$P(A,B) \leftarrow Q(A,B), R(B).$
1	*conj*	$P(A) \leftarrow Q(A), R(A).$
1	*conj2*	$P(A) \leftarrow Q(A,B), R(A,B).$
1 and 2	*chain*	$P(A,B) \leftarrow Q(A,C), R(C,B).$

Figure 2: Common metarules from the MIL literature: the letters P, Q, R denote existentially quantified second-order variables and the letters A, B, C universally quantified first-order variables.

Null Hypothesis 3: *Augmenting MIL systems with bottom-up iterations can not improve learning times.*

To test these null hypotheses, we compare the regular version of Metagol versus Metagol augmented with bottom-up iterations. We provide the two systems with the same background knowledge and metarules in each experiment. Therefore, the only variable is the learning system[1].

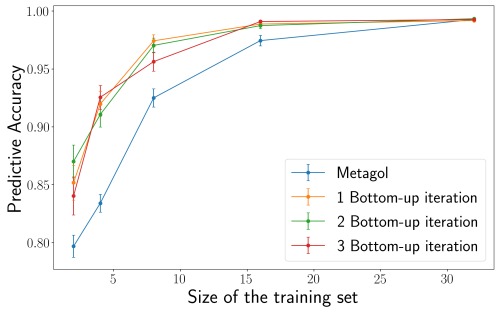
6.2 Rook Protected in Chess Endgame KRK

KRK denotes the chess ending with white having a king and a rook and black having a king. An optimal strategy is known and its correctness has been demonstrated [Bratko, 1978]. It involves reducing the area available to the black king with the white rook whilst constantly protecting the later with the white king. This experiment considers the task of learning whether the white rook is protected by its king which is an essential feature to conduct an optimal strategy.

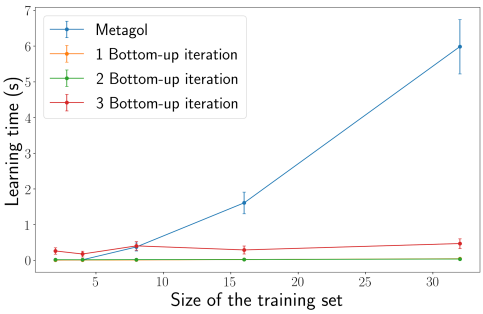
Material. The state of the board is a list of non-empty cells. Cells are atoms of the form *c(X,Y,Color,Type)* and encode the current position X/Y of the piece of color *Color* (black or white) and type *Type* (king or rook). Primitives are *piece/2*, *rook/1*, *king/1*, *white/1*, *black/1* and *distance1/2* which holds if the arguments are two pieces separated by a Chebyshev distance of 1. Metarules provided (Figure 2) belong to H^2_2. The target theory is shown in Figure 1b.

Methods. Training instances are instances of the form $rook\_protected(S)$ where S is a board state. Positive examples are generated by placing the white rook and the black king on different squares randomly selected on an empty board. The white king is placed on a random empty square at distance 1 from the rook. Negative examples are generated by altering an attribute (rank, file, color or type) selected at random of either the white king or the rook in a positive example such that the target theory does not hold for the resulting state. Training sets are built with half positive, half negative examples. We perform between 1 and 3 bottom-up iterations. Initial facts are built from a sample of size 1 of the positive examples. We measure the number of correct classifications over a set of 300 test instances generated following the same process as the training set. The default accuracy is 0.5. We compare accuracies and learning times versus the number of training examples. We measure the standard error of the mean over 100 repetitions.

Results. Accuracy results are presented in Figure 3a: a T-test suggests that the difference in accuracy is statistically significant ($p < 0.05$) for a training set up with up to 16 instances thus refuting Null Hypothesis 1. Sample complexity



(a) KRK: accuracy



(b) KRK: learning time

Accuracy	Metagol	k=1	k=2	k=3
0.85	5	2	2	3
0.9	7	4	4	4
0.95	13	7	7	8
0.98	21	13	13	14

(c) Experimental Sample Complexity vs the number of bottom-up iterations k

Figure 3: KRK results: Learning "rook protected by the king"

results are detailed in Figure 3c and show that performing bottom-up iterations can reduce the sample complexity. It is consistent with Proposition 4.1 and we thus refute null hypothesis 2. Metagol with bottom-up steps requires shorter learning times as shown in Figure 3b thus refuting null hypothesis 3. The surface hypothesis from Figure 1b has a size typically reduced from 5 to 3 clauses after $k = 1$ bottom-up iteration and to 1 clause for $k = 2$. It is better than the expected reduction of 1 and 2 clauses for $k = 1$ and $k = 2$ respectively guaranteed by Theorem 4.2. The number of predicates available after the iterations $k = 1$ and $k = 2$ typically is at most $p_1 = 18$ and $p_2 = 42$ respectively. These numbers are much lower than the worst case scenario bound provided by Theorem 4.1 which guaranteed $p_1 < 150$ and $p_2 < 90150$ and are consistent with Proposition 4.1. Performing more bottom-up iterations ($k > 2$) is not beneficial for this experiment as it can not further reduce the size of the target theory but increases learning times.

6.3 String Transformations

Materials. We consider 94 real-world string transformation problems evaluated in [Cropper, 2019] and inspired from [Lin *et al.*, 2014; Gulwani, 2011]. The dataset contains 10 positive examples for each problem. Each example is an atom of the form $task(s1, s2)$ where $task$ is the task name and $s1$ and $s2$ are input and output strings respectively. Some examples

[1]The code for reproducing the experiments is available at https://github.com/celinehocquette/bottom_up.git

Input	Output
James Brown	BROWN
David Batty	BATTY

(a) Task p9: examples

Input	Output
James Brown	JB
Joanie Faas	JF

(b) Task p39: examples

p39(A,B):-copy1skipwordskip1copy1(A,C),**skiprest**(C,B).
copy1skipwordskip1copy1(A,B) :- copy1skipword(A,C), skip1copy1(C,B).
copy1skipword(A,B):-**copy1**(A,C), **skipalphanum**(C,B).
skip1copy1(A,B):- **skip1**(A,C), **copy1**(C,B).

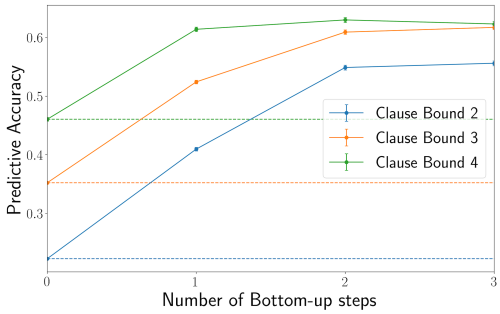
(c) Hypothesis learned for task p39: initial predicates are represented in bold. Predicates have been renamed for clarity.

Figure 4: String transformation experiment

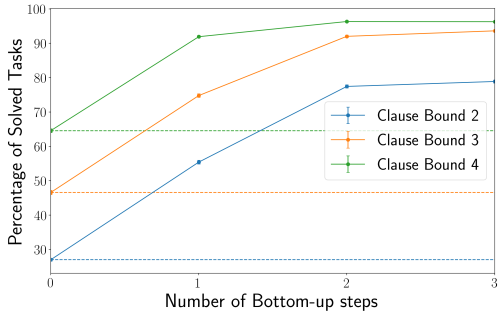
are shown in Figure 4a and 4b. For each run and each task, one example is randomly selected to form the training set, the remaining nine being being left for testing.

Methods. We provide both systems with the same background knowledge from [Lin *et al.*, 2014] containing the predicates *copyalphanum/2*, *copy1/2*, *write_point/2*, *skipalphanum/2*, *skip1/2*, *skiprest/2*, *make_lowercase/2*, *make_uppercase/2*, *make_uppercase1/2* and the *chain* metarule (Figure 2). A functional restriction is set to compensate for the lack of negative examples. We set the size of the top-down learner search space to $n \in [2, 4]$ clauses. For $k \in [0, 3]$, k bottom-up iterations are performed. If a learning tasks fails and no hypothesis is returned, the default accuracy is 0. A timeout is set to 10 minutes. We measure the standard error of the mean over 20 repetitions of the 94 tasks.

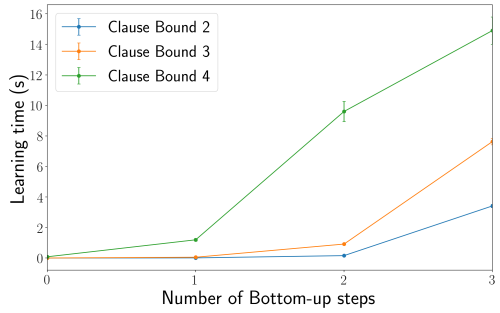
Results. Results are presented in Figure 5. The accuracy increases with the number of bottom-up iterations, outperforming the regular version of Metagol, thus refuting null hypothesis 1. Moreover, both the accuracy and the percentage of tasks solved for a clause bound of n and after k bottom-up iterations are greater than that's of Metagol for a clause bound of $n + k$. Indeed, k bottom-up iterations can reduce the target theory by more than k clauses which is more than the guarantee provided by Theorem 4.2. Learned hypotheses are characterized by a high usage of predicates invented in bottom-up iterations. For example, the hypothesis learned for the task 39 is shown in Figure 4c. Respectively one and two bottom-up iterations are enough to build the bottom two or three invented substrate predicates in which case the surface hypothesis has respectively 2 and one clauses. Conversely, Metagol requires 4 clauses to learn the same hypothesis.



(a) Accuracy



(b) Percentage of tasks solved



(c) Learning time of solved tasks

Figure 5: String transformation results: baselines (dotted lines) correspond to the regular version of Metagol, results for Metagol augmented with bottom-up steps are represented with solid lines

7 Conclusion

We have introduced in this work a method for performing extensive predicate invention in a bottom-up fashion within MIL. The background knowledge is generalised with an extension of the immediate consequence operator for second-order logic programs. We have demonstrated this method is complete with respect to a fragment of dyadic datalog. This methods reduces the number of clauses to be learned by the top-down learner, which can reduce the sample complexity. Our experimental results provide convincing evidence to support this claim.

We have restricted the scope of this work to second-order metarules. The use of higher-order metarules [Cropper and Muggleton, 2016a] could allow higher-order definitions to be learned, thus allowing disjunctions or recursions to be built in bottom-up iterations.

Preliminary experiments suggest our method is more effective for target theories for which the calling diagram has a tree structure instead of a more complex directed cyclic or acyclic graph. We intend to conduct a theoretical characterisation of target theories for which this method is more effective. This characterisation will be based upon the degree of reuse of invented predicates in the calling diagram and the density of relevant predicates in the search space. We also intend to determine theoretically the optimal number of bottom-up iterations given a search space.

We have shown a theoretical condition over the number of predicate symbols introduced which guarantees a sample complexity gain. We will devise a more discriminative selection of relevant predicates to guarantee this condition is fulfilled which would provide a better scalability of this method.

References

[Bratko, 1978] Ivan Bratko. Proving correctness of strategies in the AL1 assertional language. *Information Processing Letters*, 7(5):223 – 230, 1978.

[Corapi *et al.*, 2011] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive logic programming in answer set programming. In *ILP*, volume 7207 of *Lecture Notes in Computer Science*, pages 91–97. Springer, 2011.

[Cropper and Muggleton, 2016a] Andrew Cropper and Stephen H. Muggleton. Learning Higher-Order Logic Programs through Abstraction and Invention. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1418–1424. IJCAI/AAAI Press, 2016.

[Cropper and Muggleton, 2016b] Andrew Cropper and Stephen H. Muggleton. Metagol system. https://github.com/metagol/metagol, 2016.

[Cropper, 2019] Andrew Cropper. Playgol: Learning Programs Through Play. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 6074–6080. ijcai.org, 2019.

[Fensel and Wiese, 1993] Dieter Fensel and Markus Wiese. Refinement of rule sets with jojo. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, pages 378–383, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[Gulwani, 2011] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM.

[Law *et al.*, 2014] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In E. Fermé and J. Leite, editors, *Logics in Artificial Intelligence*, pages 311–325, Cham, 2014. Springer International Publishing.

[Lin *et al.*, 2014] Dianhuan Lin, Eyal Dechter, Kevin Ellis, Joshua B. Tenenbaum, and Stephen H. Muggleton. Bias reformulation for one-shot function induction. *In Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014)*, pages 525–530, 2014.

[Lloyd, 1984] John W. Lloyd. *Foundations of Logic Programming, 1st Edition*. Springer, 1984.

[Maher, 1988] Michael J. Maher. Chapter 16 - equivalences of logic programs. pages 627 – 658, 1988.

[Mitchell, 1982] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203 – 226, 1982.

[Muggleton and Buntine, 1988] Stephen H. Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In John Laird, editor, *Machine Learning Proceedings 1988*, pages 339 – 352. Morgan Kaufmann, San Francisco (CA), 1988.

[Muggleton and Lin, 2013] Stephen H. Muggleton and Dianhuan Lin. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *In Proceedings of the 23rd International Joint Conference Artificial Intelligence*, pages 1551–1557, 2013.

[Muggleton *et al.*, 2012] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter A. Flach, Katsumi Inoue, and Ashwin Srinivasan. ILP turns 20. *Machine Learning*, 86(1):3–23, Jan 2012.

[Muggleton *et al.*, 2014] Stephen H. Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning 94*, pages 25–49, 2014.

[Muggleton, 1991] Stephen H. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, Feb 1991.

[Muggleton, 1995] Stephen H. Muggleton. Inverse Entailment and Progol. *New Generation Comput.*, 13(3&4):245–286, 1995.

[Quinlan, 1983] J. Ross Quinlan. *Learning Efficient Classification Procedures and Their Application to Chess End Games*, pages 463–482. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[Rabinovich *et al.*, 1996] Savely Rabinovich, Gregory Berkolaiko, and Shlomo Havlin. Solving nonlinear recursions. *Journal of Mathematical Physics*, 37:5828–5836, November 1996.

[Shapiro and Niblett, 1982] Alen D. Shapiro and Tim Niblett. Automatic induction of classification rules for a chess endgame. In M.R.B. Clarke, editor, *Advances in Computer Chess*, volume 3, pages 73–91. Pergammon, Oxford, 1982.

[Stahl, 1993] Irene Stahl. Predicate invention in ILP — an overview. In P. B. Brazdil, editor, *Machine Learning: ECML-93*, pages 311–322, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[Stahl, 1995] Irene Stahl. The appropriateness of predicate invention as bias shift operation in ILP. *Machine Learning*, 20(1):95–117, Jul 1995.

[Van Emden and Kowalski, 1976] Maarten H. Van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, October 1976.

[Zelle *et al.*, 1994] John M. Zelle, Raymond J. Mooney, and Joshua B. Konvisser. Combining top-down and bottom-up techniques in inductive logic programming. In William W. Cohen and Haym Hirsh, editors, *Machine Learning Proceedings 1994*, pages 343 – 351. Morgan Kaufmann, San Francisco (CA), 1994.