Meta-Interpretive Learning Using HEX-Programs*

Tobias Kaminski^1 , **Thomas Eiter**^{1} and **Katsumi Inoue**^{2}
^{1}Technical University of Vienna (TU Wien), Vienna, Austria
^{2}National Institute of Informatics, Tokyo, Japan
{kaminski,eiter}@kr.tuwien.ac.at, inoue@nii.ac.jp

Abstract

Meta-Interpretive Learning (MIL) is a recent approach for Inductive Logic Programming (ILP) implemented in Prolog. Alternatively, MIL-problems can be solved by using Answer Set Programming (ASP), which may result in performance gains due to efficient conflict propagation. However, a straightforward MIL-encoding results in a huge size of the ground program and search space. To address these challenges, we encode MIL in the HEX-extension of ASP, which mitigates grounding issues, and we develop novel pruning techniques.

1 Introduction

Meta-Interpretive Learning (*MIL*) learns definite logic programs from *positive* and *negative examples* wrt. *background knowledge* (*BK*) by instantiating so-called *meta-rules* [Muggleton *et al.*, 2015]. MIL is very powerful as it enables *predicate invention*, i.e. to use new predicates during learning, and supports induction of recursive programs, while the hypothesis space is constrained effectively by meta-rules. *Metagol* [Cropper and Muggleton, 2016b] is an implementation of MIL based on a *Prolog* meta-interpreter. The system is very efficient by exploiting the query-driven procedure of Prolog.

While traditionally, most ILP systems are based on Prolog, the advantages of *Answer Set Programming* (*ASP*) [Gelfond and Lifschitz, 1991] for ILP were recognized and several ASP-based systems have been developed, e.g. [Otero, 2001; Ray, 2009; Law *et al.*, 2014]. One advantage of these systems is that they can leverage the efficiency of modern ASP-solvers such as CLASP [Gebser *et al.*, 2012], which supports conflict propagation and learning. Muggleton et al. [2014] already considered an ASP-encoding of MIL, which used only one specific meta-rule and was tailored to inducing grammars. They observed that ASP can have an advantage over Prolog due to effective pruning, but that it performs worse when the BK is extensive or there are few constraints.

Implementing general MIL by ASP comes with its own challenges; and solving a MIL-problem $\mathcal{M}$ efficiently by using a straightforward ASP-encoding is often infeasible. The first challenge is the large search space as a result of an unguided search and lack of procedural bias. The second and more severe challenge concerns the *grounding bottleneck* of ASP: in contrast to Prolog, where only relevant terms are regarded by *unification*, all terms that possibly occur in a derivation must be considered in a grounding step. Finally, a third challenge are recursive manipulations of structured objects, such as lists, that are common in Prolog, but less supported in ASP.

We meet the mentioned challenges for a class of MIL-problems widely encountered in practice, by developing an MIL-encoding in the HEX-formalism [Eiter *et al.*, 2016]. HEX-programs extend ASP with a bidirectional information exchange between a program and arbitrary external computation sources. Our main contributions are the following:

- We introduce a general MIL-encoding $\Pi(\mathcal{M})$, where we restrict the search space by interleaving derivations at the object and the meta level; and we outsource the BK using HEX to enable the manipulation of complex objects.
- We define the class of *forward-chained* MIL-problems, for which the grounding can be restricted by guarding the import of new terms from the BK in a second encoding, $\Pi_f(\mathcal{M})$, that imports terms in an inductive manner.
- In addition, we develop a technique to abstract from BK-constants in a third encoding, $\Pi_{sa}(\mathcal{M})$, by precomputing sequences of BK-atoms that derive positive examples and by checking negative examples externally.
- We present empirical results, which provide evidence for the potential of using a HEX-based approach for MIL.

2 Background

We assume familiarity with basic concepts of ASP [Eiter *et al.*, 2009] (for further background, cf. [Muggleton *et al.*, 2015; Eiter *et al.*, 2016]).

Meta-Interpretive Learning. MIL utilizes higher-order *meta-rules* to declaratively constrain a hypothesis space. We focus on meta-rules of the form $P(x,y) \leftarrow Q_1(x_1,y_1),...,Q_k(x_k,y_k),R_1(z_1),...,R_n(z_n)$, where P, Q_i, $1 \leq i \leq k$, and R_j, $1 \leq j \leq n$, are higher-order variables, and x,y,x_i,y_i, $1 \leq i \leq k$, and z_j, $1 \leq j \leq n$, are first-order variables

*This is an abridged version of a paper presented at ICLP'18 [Kaminski et al., 2018a]. The research has been supported by the Austrian Science Fund (FWF) projects P27730 and W1255-N23. Tobias Kaminski has also been supported by the NII International Internship Program and the JSPS KAKENHI Grant JP17H00763.

s.t. x and y also occur in the body. Examples of concrete meta-rules are the "Precon"-rule $P(x,y) \leftarrow Q(x), R(x,y)$ and the "Chain"-rule $P(x,y) \leftarrow Q(x,z), R(z,y)$. A *meta-substitution* of a meta-rule R is an instantiation of R where all higher-order variables are substituted by predicate symbols.

Definition 1 *A* Meta-Interpretive Learning *(*MIL-*)*problem *is a quadruple $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$, where B is a definite program (*BK*); E^+ and E^- are finite sets of binary ground atoms (*positive *resp.* negative examples*); and $\mathcal{R}$ is a finite set of meta-rules. We say that B is* extensional *if it contains only ground atoms. A* solution *for $\mathcal{M}$ is a* hypothesis *$\mathcal{H}$ consisting of a set of meta-substitutions of meta-rules in $\mathcal{R}$ s.t. $B \cup \mathcal{H} \models e^+$ for each $e^+ \in E^+$ and $B \cup \mathcal{H} \not\models e^-$ for each $e^- \in E^-$.*

To obtain hypotheses that generalize well, Metagol computes a solution containing a minimal number of rules.

Example 1 *Consider the MIL-problem $\mathcal{M}$ with*

- $B=\{m(an,jo), f(bob,jo), m(sue,an), f(tim,an)\}$,
- $E^+=\{a(sue,jo), a(tim,jo), a(bob,jo)\}$,
- $E^-=\{a(jo,tim)\}$, *and*
- $\mathcal{R}=\{P(x,y) \leftarrow Q(x,y); P(x,y) \leftarrow Q(x,z), R(z,y)\}$,

abbreviating $\underline{m}$other, $\underline{f}$ather and $\underline{a}$ncestor. A minimal solution for $\mathcal{M}$ is $\{p1(x,y) \leftarrow f(x,y);\ p1(x,y) \leftarrow m(x,y);\ a(x,y) \leftarrow p1(x,y);\ a(x,y) \leftarrow p1(x,z), a(z,y)\}$, where $p1$ is an invented predicate intuitively representing 'parent'.

HEX**-Programs**. HEX-programs extend disjunctive logic programs by *external atoms* in rule bodies. Ground external atoms are of form $\&g[\mathbf{p}](\mathbf{c})$, where $\mathbf{p} = p_1, ..., p_k$ are predicates or constants, called *input parameters*, and $\mathbf{c} = c_1, ..., c_l$, are constant *outputs*. The semantics of $\&g[\mathbf{p}](\mathbf{c})$ wrt. an interpretation I is determined by a $1{+}k{+}l$-ary (*Boolean*) *oracle function* $f_{\&g}$ such that $I \models \&g[\mathbf{p}](\mathbf{c})$ iff $f_{\&g}(I, \mathbf{p}, \mathbf{c}) = 1$. In practice, oracle functions are realized as solver-plugins. The answer sets of a ground HEX-program Π are those interpretations I over ordinary atoms which are minimal models of the program consisting of all instances of rules in Π s.t. I satisfied their bodies (the so-called *FLP-reduct* [Faber *et al.*, 2011]).

Example 2 *Let us consider the* HEX*-program $\Pi = \{l([a,a]);\ l(y) \leftarrow \&remove[x](y), l(x)\}$, where $f_{\&remove}(I, X, Y) = 1$ iff X and Y are ground lists and Y is X without the first element. The single answer set of Π is $\{l([a,a]), l([a]), l([])\}$.*

In slight abuse of notation, we technically regard complex ground terms such as $[a,a]$ as constants in our approach. As external atoms can have predicate inputs, their semantics may also depend on the extension of predicates in an answer set.

3 HEX-Encoding of MIL

A main motivation for applying ASP for MIL is that constraints wrt. negative examples can be propagated by an ASP-solver, while Metagol checks them only at the end. For this, ordinary ASP is sufficient, but we employ HEX as it also allows to outsource the BK from an encoding. This enables to limit the import of BK and to specify intensional BK using, e.g., string or list operations (usually not available in ASP).

As we consider meta-rules using unary and binary atoms, we introduce external atoms for importing the relevant unary and binary atoms that are entailed by the BK in an encoding.

Definition 2 *Given an MIL-problem $\mathcal{M}$, we call the external atom $\&bkUnary[ded](X,Y)$* unary BK-atom *and $\&bkBinary[ded](X,Y,Z)$* binary BK-atom*, where $f_{\&bkUnary}(I, ded, X, Y) = 1$ iff $B \cup \{p(a,b) \mid ded(p,a,b) \in I\} \models X(Y)$, and $f_{\&bkBinary}(I, ded, X, Y, Z) = 1$ iff $B \cup \{p(a,b) \mid ded(p,a,b) \in I\} \models X(Y,Z)$.*

The BK-atoms receive as input the extension of the predicate ded, which represents the set of all atoms deducible from a candidate hypothesis. Their output constants represent atoms entailed by the BK plus the atoms of ded.

In theory, MIL can be encoded by applying the well-known *guess-and-check* methodology, i.e. by generating all combinations of substitutions of the given meta-rules and available predicate symbols, deriving all entailed atoms, and checking compatibility with examples using constraints. However, usually a large fraction of rules is irrelevant for inducing a hypothesis as they can never fire. Thus, we interleave guesses on the meta level and derivations on the object level, ensuring that rules are only added if their bodies are already deducible.

We associate each meta-rule R with a unique identifier R_{id} and a set of *ordering constraints*, $R_{ord} \subseteq \{ord(P,Q) \mid P, Q$ are higher-order variables that occur in $R\}$, and assume a predefined total ordering $\succeq_{\mathcal{P}}$ over all predicates. By $\mathcal{S}$ we denote a finite set of fresh *Skolem predicates* usable for predicate invention. Our general encoding is then defined as follows.

Definition 3 *Given an MIL-problem $\mathcal{M} = (B, E^+, E^-, \mathcal{R})$, let Sig be the set containing all $p \in \mathcal{S}$ and each predicate p that occurs in $E^+ \cup E^-$ or in a rule head in B. The* HEX-MIL-encoding *for $\mathcal{M}$ is the* HEX*-program $\Pi(\mathcal{M})$ containing*

(1) $sig(p) \leftarrow$ *for each* $p \in Sig$, *and*
$ord(p,q) \leftarrow$ *for all* $p, q \in Sig$ *s.t.* $p \succeq_{\mathcal{P}} q$

(2) $unary(x,y) \leftarrow \&bkUnary[ded](x,y)$ *and*
$ded(x,y,z) \leftarrow \&bkBinary[ded](x,y,z)$

(3) for each meta-rule $R = P(x,y) \leftarrow Q_1(x_1,y_1), ..., Q_k(x_k,y_k), R_1(z_1), ..., R_n(z_n) \in \mathcal{R}$ *and* $\{ord(P,Q_{i_1}), ..., ord(P,Q_{i_m})\} = \{ord(P,Q_i) \in R_{ord} \mid 1 \le i \le k\}$*:*

(a) $\{meta(R_{id}, x_P, x_{Q_1}, ..., x_{Q_k}, x_{R_1}, ..., x_{R_n})\} \leftarrow sig(x_P), sig(x_{Q_1}), ..., sig(x_{Q_k}), sig(x_{R_q}), ..., sig(x_{R_n}), ord(x_P, x_{Q_{i_1}}), ..., ord(x_P, x_{Q_{i_m}}), ded(x_{Q_1}, x_1, y_1), ..., ded(x_{Q_k}, x_k, y_k), unary(x_{R_1}, z_1), ..., unary(x_{R_n}, z_n)$

(b) $ded(x_P, x, y) \leftarrow meta(R_{id}, x_P, x_{Q_1}, ..., x_{Q_k}, x_{R_1}, ..., x_{R_n}), ded(x_{Q_1}, x_1, y_1), ..., ded(x_{Q_k}, x_k, y_k), unary(x_{R_1}, z_1), ..., unary(x_{R_n}, z_n)$

(4) $\leftarrow$ not $ded(p,a,b)$ *for each* $p(a,b) \in E^+$, *and*
$\leftarrow ded(p,a,b)$ *for each* $p(a,b) \in E^-$.

The predicate $meta$ contains meta-substitutions added to a hypothesis, and ded captures all atoms that can be deduced from a guessed hypothesis. Item (3) contains the meta-level guessing part (a) and the object-level deduction part (b). A meta-substitution can be guessed to be in a solution only if first-order instantiations of its body atoms can be deduced. Substituted predicates must be from the signature Sig and the ordering constraints must be satisfied as stated by the facts in item (1). The BK is imported in item (2), and item (4) adds the constraints imposed by positive and negative examples.

Solutions can be obtained from the answer sets of $\Pi(\mathcal{M})$:

Definition 4 *For a set of meta-rules $\mathcal{R}$, the* logic program induced by *an interpretation I consists of all rules obtained from atoms $meta(R_{id}, x_P, x_{Q_1}, ..., x_{Q_k}, x_{R_1}, ..., x_{R_n}) \in I$, where $R = P(x,y) \leftarrow Q_1(x_1,y_1), ..., Q_k(x_k,y_k), R_1(z_1), ..., R_n(z_n) \in \mathcal{R}$, by substituting P by x_P, Q_i by x_{Q_i} for $1 \leq i \leq k$, and R_j by x_{R_j} for $1 \leq j \leq n$.*

Every answer set of $\Pi(\mathcal{M})$ encodes a solution, and all solutions $\mathcal{H}$ that only contain *productive* rules, i.e. rules s.t. all atoms in the body of some ground instance is entailed by $B \cup \mathcal{H}$, can be generated in this way.

Theorem 1 *Given an MIL-problem $\mathcal{M}$, (i) if S is an answer set of $\Pi(\mathcal{M})$, the logic program $\mathcal{H}$ induced by S is a solution for $\mathcal{M}$; and (ii) if $\mathcal{H}$ is a solution for $\mathcal{M}$ s.t. all rules in $\mathcal{H}$ satisfy R_{ord} and are productive, then there is an answer set S of $\Pi(\mathcal{M})$ s.t. $\mathcal{H}$ is the logic program induced by S.*

While $\Pi(\mathcal{M})$ performs well when only few constants are introduced by the BK, the grounding becomes prohibitively large when more constants are imported. We thus introduce a class of MIL-problems for which the import can be restricted.

Definition 5 *A* forward-chained MIL-problem *only contains meta-rules of the form $P(z_0, z_k) \leftarrow Q_1(z_0, z_1), \ldots, Q_i(z_{i-1}, z_i), \ldots, Q_k(z_{k-1}, z_k)$, $R_1(x_1), \ldots, R_l(x_l)$, where $1 \leq i \leq k$, $0 \leq l$, and $x_j \in \{z_0, \ldots, z_k\}$ for all $1 \leq j \leq l$.*

Intuitively, all first-order variables in the body of such meta-rules are part of a chain between the first and second argument in the head. Viewing binary BK-predicates as mappings from their first to their second argument, only atoms from an extensional BK are relevant that occur in a chain between the first and second argument of examples. To restrict the import of BK, we modify the external atoms from Def. 2 such that their outputs are guarded by an input constant.

Definition 6 *For a forward-chained MIL-problem $\mathcal{M}$ with extensional B, the external atom &fcUnary$[Y](X)$ (resp. &fcBinary$[Y](X,Z)$) is a* unary *(resp.* binary*)* forward-chained BK-atom, *if $f_{\&fcUnary}(I,Y,X) = 1$ iff $X(Y) \in B$ (resp. $f_{\&fcBinary}(I,Y,X,Z) = 1$ iff $X(Y,Z) \in B$).*

As we assume an extensional BK, the input parameter ded is not needed for forward-chained BK-atoms. Based on the previous definition, we can modify our HEX-MIL-encoding such that only relevant atoms from the BK are imported.

Definition 7 *Given a forward-chained MIL-problem $\mathcal{M}$ where B is extensional, the* forward-chained HEX-MIL-encoding *for $\mathcal{M}$ is the* HEX*-program $\Pi_f(\mathcal{M})$ containing items (1), (3) and (4) from Definition 3, and the rules*

(f1) $unary(x,y) \leftarrow$ &fcUnary$[y](x), s(y)$

(f2) $ded(x,y,z) \leftarrow$ &fcBinary$[y](x,z), s(y)$

(f3) $s(a) \leftarrow$ *for each* $p(a,b) \in E^+ \cup E^-$

(f4) $s(y) \leftarrow ded(\_, \_, y)$

The main difference between $\Pi_f(\mathcal{M})$ and $\Pi(\mathcal{M})$ is that the import of BK is guarded by the predicate s in (f1) and (f2), whose extension contains all constants appearing as first argument of an example, due to (f3), and all constants that appear in deductions from already imported BK, due to (f4).

Every answer set of $\Pi_f(\mathcal{M})$ still corresponds to a solution, but not all solutions may be obtained. Nonetheless, it is ensured that a minimal solution (i.e., with fewest meta-substitutions) is encoded by some answer set:

Theorem 2 *Let $\mathcal{M}$ be a forward-chained MIL-problem with extensional B. Then, (i) for every answer set S of $\Pi_f(\mathcal{M})$, the logic program induced by S is a solution for $\mathcal{M}$; and (ii) there is an answer set S' of $\Pi_f(\mathcal{M})$ s.t. the logic program induced by S' is a minimal solution for $\mathcal{M}$ if one exists.*

As in practice, we employ iterative deepening search for computing a minimal solution, one is guaranteed to be found.

4 State Abstraction

Based on the observation that binary BK-predicates can be applied sequentially when MIL-problems are forward-chained, we now discuss a technique that eliminates object-level constants from an encoding entirely. For brevity, we omit formal definitions, which can be found in [Kaminski *et al.*, 2018a], and illustrate the core idea by an example.

Example 3 *Consider $\mathcal{M}$ where B contains the extensional BK represented by the facts $remove([X|R], R) \leftarrow$, $switch([X,Y|R],[Y,X|R]) \leftarrow$, and $firstA([a|R]) \leftarrow$. Let $E^+ = \{p([c,a,b,a],[c])\}$, $E^- = \{p([c,b,a,b],[c])\}$, and $\mathcal{R} = \{P(x,y) \leftarrow Q(x,z), R(z,y);\ P(x,y) \leftarrow Q(x,y), R(y);\ P(x,y) \leftarrow Q(x,y)\}$. Intuitively, a solution needs to memorize c and delete the rest if the input list has a at position 2; this requires to repeatedly switch the first two elements and to remove the first. A solution is $\mathcal{H} = \{p(x,y) \leftarrow p1(x,z), p(z,y);\ p(x,y) \leftarrow remove(x,y);\ p1(x,y) \leftarrow switch(x,y), firstA(y);\ p1(x,y) \leftarrow remove(x,z), switch(z,y)\}$. In fact, any program that enables derivations alternating between the actions $switch$ and $remove$ and prevents the derivation of the negative example using $firstA$ as guard is a solution.*

Note that here, finding a correct sequence for derving a positive example can be viewed as a *planning problem*, where object-level constants represent *states*, binary BK-predicates are viewed as *actions*, and unary BK-predicates are *fluents*. Our *state abstraction* technique exploits the insight that the tasks of (1) solving the planning problem and (2) finding a matching hypothesis can be separated, where the HEX-program encodes task (2), and computations involving states are performed externally. We represent possible plans to derive positive examples by sequences of binary BK-atoms.

Example 4 (cont'd) *The positive example in Example 3 can be derived by the sequence $s([c,a,b,a],[a,c,b,a])$, $r([a,c,b,a],[c,b,a])$, $s([c,b,a],[b,c,a])$, $r([b,c,a],[c,a])$, $s([c,a],[a,c])$, $r([a,c],[c])$, where s = $switch$, r = $remove$.*

To import action sequences that derive positive examples and fluents that hold in states, we use two external atoms, &saUnary$[](X,Y)$ and &saBinary$[](X,Y,Z)$. States are represented by integers as the concrete constants are irrelevant for generalizing sequences. For our example, e.g. &saBinary$[](switch, (e^+_{id}, seq_{id}, 1), (e^+_{id}, seq_{id}, 2))$ is true, where e^+_{id} identifies the positive example, seq_{id} identifies the sequence from Ex. 4, and the integers 1 and 2 represent the states $[c,a,b,a,b]$ and $[a,c,b,a,b]$, resp., where the second state can be reached from the first state by applying the action $switch$. We use a further external atom

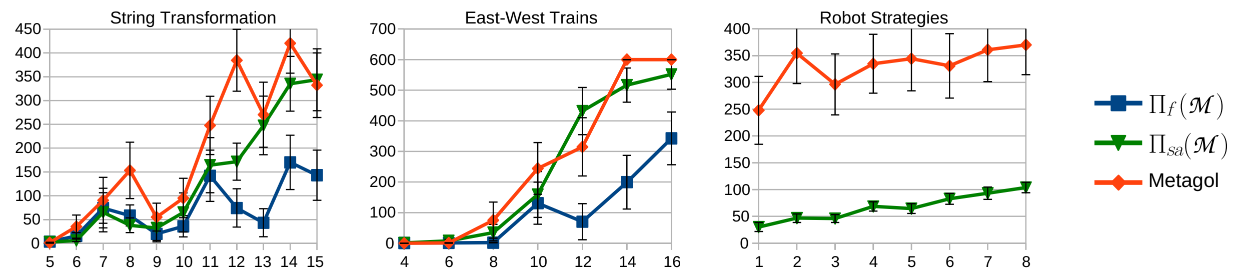

Figure 1: Results of experiments (B1)-(B3). Average runtimes in seconds are shown on the y-axis, and instance sizes on the x-axis.

$\&checkPos[](X_1, X_2, Y, Z)$ to retrieve the start and end states of sequences associated with positive examples, based on which one sequence is selected for each positive example.

Finally, the check for non-derivability of negative examples cannot be performed in an encoding without importing BK-constants. Hence, we also realize this check by an external atom $\&failNeg[meta]()$ used in a constraint, which evaluates to true as soon as a negative example is derivable.

Example 5 *Consider* $\mathcal{M}$ *with* $B = \{q(a,b),\ q(a,c), r(a,b)\}$, $E^+ = \{p(a,b)\}$, $E^- = \{p(a,c)\}$, *and* $\mathcal{R} = \{R = P(x,y) \leftarrow Q(x,y)\}$. *For* $I = \{meta(R_{id}, p, q)\}$, *we obtain* $f_{\&failNeg}(I, meta) = 1$ *as the negative example can be derived from* $B \cup \{p(x,y) \leftarrow q(x,y)\}$*; a solver can exploit the fact that* $p(x,y) \leftarrow q(x,y)$ *cannot belong to any solution.*

Utilizing the external atoms from above, we define an encoding $\Pi_{sa}(\mathcal{M})$ that separates the planning from the generalization problem and imports no BK-constants. It can be shown that $\Pi_{sa}(\mathcal{M})$ yields correct solutions, and a minimal one if all sequences that derive positive examples are acyclic.

5 Empirical Evaluation

For experimentation, we used the *hexlite* solver 0.3.20, and *SWI-Prolog* 7.2.3 to run Metagol 2.2.0. Tests were run on a 2.5 GHz Intel Core i5 processor and 8 GB RAM; the timeout was 600 secs per instance. The results wrt. the average runtimes over 20 instances per size (for B1 and B3), resp. 10 (for B2), in secs are shown in Fig. 1 (error bars show the *standard error*). We compared $\Pi_f(\mathcal{M})$ and $\Pi_{sa}(\mathcal{M})$ to Metagol.

String Transformation (B1). This benchmark is based on Ex. 3, and akin to inducing *regular grammars* as in [Muggleton *et al.*, 2014], but we also allow switching the first two letters in a string in addition to removing elements. This increases the search space and makes conflict propagation and state abstraction more relevant. We used positive and negative examples of form $p([c|X],[c])$, where X is a random sequence of letters a and b. The BK-predicates are *remove*, *switch*, *firstA*, *firstB* and *firstC*. We used one positive and one negative example of same length $n \in [1, 15]$.

East-West Trains (B2). The *east-west train challenge* is a popular ILP-benchmark [Larson and Michalski, 1977]. The task is to classify trains based on features (e.g. shapes of cars and types of loads) to be east- or westbound. We used eastbound trains as positive and westbound trains as negative examples. The BK defines the predicate *removeCar* which removes the first car from a train; and we declare 50 unary predicates, e.g. *shape_rectangle* or *load_triangles*, for checking features of the remaining part of a train. Our data set comprised 10 east- and 10 westbound trains [Michie *et al.*, 1994] as used previously [Muggleton *et al.*, 2015]. We generated instances of size $n \in \{4, 6, 8, 10, 12, 14, 16\}$ by randomly selecting n from the 20 trains, s.t. $n/2$ were eastbound.

Robot Strategies (B3). Finally, we considered learning robot strategies as in [Cropper and Muggleton, 2016a]: customers sit at a table and a waiter robot needs to serve each customer her desired drink. Positive examples map initial states to goal states for varying numbers of customers and drink preferences. We used the actions *move_right*, *pour_coffee* and *pour_tea*, and the fluents *wants_coffee*, *wants_tea* and *at_end*. Negative examples are implicitly given by all binary atoms that map an initial state to a non-goal state. We generated random instances similar to [Cropper and Muggleton, 2016a], where each positive example has a random number of $i \in [1, 10]$ customers with random preferences, and the instance size is the number of positive examples $n \in [1, 8]$.

Findings. Regarding (B1) and (B2), we found that instances can be solved significantly faster by employing $\Pi_f(\mathcal{M})$ than by Metagol due to conflict propagation in ASP. The encoding $\Pi_{sa}(\mathcal{M})$ performed similar to Metagol as only two binary predicates, resp. one predicate, are defined by the BK s.t. solving the planning problem externally yields not a big advantage, and the benefit of conflict propagation is outweighed by the overhead that goes along with outsourcing checks for negative examples. For (B3), we did not obtain results using $\Pi_f(\mathcal{M})$ for many instances as the grounding was too large. This problem could be avoided by using state abstractions, which yielded a significant speed-up compared to Metagol.

6 Conclusion

Our approach combines several advantages of Metagol and ASP-based approaches, and it is very flexible as it allows to plug in arbitrary (monotonic) theories as BK. The potential of an ASP-based approach for MIL is supported by our experiments. There are alternative ways to encode MIL in HEX, which might work better in other contexts, e.g. when an MIL-problem has no negative examples. For instance, while the encodings in this paper derive object-level facts in a bottom-up fashion, a more directed top-down search can also be simulated in ASP [Kaminski *et al.*, 2018b].

References

[Cropper and Muggleton, 2016a] Andrew Cropper and Stephen H. Muggleton. Learning higher-order logic programs through abstraction and invention. In *IJCAI 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.

[Eiter *et al.*, 2009] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web 2009, Tutorial Lectures*, volume 5689 of *LNCS*, pages 40–110. Springer, 2009.

[Eiter *et al.*, 2016] Thomas Eiter, Michael Fink, Giovambattista Ianni, Thomas Krennwallner, Christoph Redl, and Peter Schüller. A model building framework for answer set programming with external computations. *TPLP*, 16(4):418–464, 2016.

[Faber *et al.*, 2011] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, 175(1):278–298, 2011.

[Gebser *et al.*, 2012] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187-188:52–89, August 2012.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–386, 1991.

[Kaminski *et al.*, 2018a] Tobias Kaminski, Thomas Eiter, and Katsumi Inoue. Exploiting answer set programming with external sources for meta-interpretive learning. *TPLP*, 18(3-4):571–588, 2018.

[Kaminski *et al.*, 2018b] Tobias Kaminski, Thomas Eiter, and Katsumi Inoue. Efficiently encoding meta-interpretive learning by answer set programming (work in progress). http://ilp2018.unife.it/wp-content/uploads/2018/08/Efficiently-Encoding-Meta-Interpretive-Learning-by-Answer-Set-Programming.pdf, 2018.

[Larson and Michalski, 1977] J. Larson and Ryszard S. Michalski. Inductive inference of VL decision rules. *SIGART Newsletter*, 63:38–44, 1977.

[Law *et al.*, 2014] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs. In *JELIA 2014*, volume 8761 of *LNCS*, pages 311–325. Springer, 2014.

[Michie *et al.*, 1994] Donald Michie, Stephen Muggleton, David Page, and Ashwin Srinivasan. To the international computing community: A new east-west challenge. Technical report, Oxford University Computing laboratory, UK, 1994.

[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(1):25–49, 2014.

[Muggleton *et al.*, 2015] Stephen H. Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.

[Otero, 2001] Ramón P. Otero. Induction of stable models. In *ILP 2001*, volume 2157 of *LNCS*, pages 193–205. Springer, 2001.

[Ray, 2009] Oliver Ray. Nonmonotonic abductive inductive learning. *J. Applied Logic*, 7(3):329–340, 2009.