Online appendix for the paper

Exploiting Answer Set Programming with External Sources for Meta-Interpretive Learning

published in Theory and Practice of Logic Programming

TOBIAS KAMINSKI, THOMAS EITER
Technical University of Vienna (TU Wien), Vienna, Austria
(*e-mail:* {kaminski,eiter}@kr.tuwien.ac.at)

KATSUMI INOUE
National Institute of Informatics, Tokyo, Japan
(*e-mail:* inoue@nii.ac.jp)

Appendix A Proof Sketches

Proof Sketch for Theorem 1
(i) The program $\mathscr{H}$ must be a solution for $\mathscr{M}$ according to Definition 1 because the constraints in item (4) of Definition 3 ensure that every positive example $e^+ \in E^+$ is derivable by rules generated by the BK imported in item (2), the rules generated by item (3b) and meta-substitutions in S obtained from guesses added by item (3a), and that no negative example $e^- \in E^-$ is derivable. In addition, atoms not imported from the BK by item (2) are not relevant for deriving examples according to Definition 2 as they are not entailed by $B \cup \mathscr{H}$. Moreover, the facts generated by item (1) are only used in the positive bodies of guessing rules generated by item (3a) such that they only constrain the guesses for meta-substitutions.

(ii) We know that $\mathscr{H}$ is a solution for $\mathscr{M}$ s.t. all meta-substitutions in $\mathscr{H}$ satisfy the respective ordering constraints and are productive. Let I be the interpretation containing all atoms corresponding to facts generated by item (1) of Definition 3 wrt. $\mathscr{M}$, the atoms $unary(p,a)$ and $deduced(p,a,b)$ for all $p(a)$ and $p(a,b)$, resp., s.t. $B \cup \mathscr{H} \models p(a)$ and $B \cup \mathscr{H} \models p(a,b)$, and the atom $meta(R_{id}, p, q_1, ..., q_k, r_1, ..., r_n)$ for every rule $p(x,y) \leftarrow q_1(x_1,y_1), ..., q_k(x_k,y_k), r_1(z_1), ..., r_n(z_n) \in \mathscr{H}$ that is a meta-substitution of the meta-rule R. Furthermore, let I' be the interpretation containing all atoms $n\_meta(R_{id}, p, q_1, ..., q_k, r_1, ..., r_n)$ that occur in a ground rule obtained from a rule generated by item (3b) whose body is satisfied by I, s.t. $meta(R_{id}, p, q_1, ..., q_k, r_1, ..., r_n) \notin I$. It can be shown that $I \cup I'$ is an answer set of $\Pi(\mathscr{M})$ s.t. $\mathscr{H}$ is the logic program induced by S. $\square$

Proof Sketch for Theorem 2
(i) Compared to the general HEX-MIL-encoding of Definition 3, only item (2) is changed by the encoding $\Pi_f(\mathscr{M})$ such that the import of BK is guarded by the predicate s. As before, item (4) ensures that every positive example $e^+ \in E^+$ is derivable. It is only left to show that if a negative example $e^- \in E^-$ is entailed by $B \cup \mathscr{H}$, then it can also be derived wrt. the BK imported via items (f1) and (f2) of Definition 7. Since all meta-rules are assumed to be forward-chained, only meta-substitutions of the form $p(z_0,z_k) \leftarrow p(x,y), \ldots, p_1(x_1,y_1), \ldots, p_k(x_k,y_k), r_1(x_1), \ldots, r_l(x_l)$ are usable for deriving examples in which x is connected to y by a chain of atoms $p_i(x_i,y_i)$ in

the body, where $y_i = x_{i+1}$, for $1 \leq i \leq k-1$, $x = x_1$ and $y = y_k$. Furthermore, (f2) imports every binary atom in the BK where the first argument already occurs as first argument in an example or as second argument in an atom previously imported from the BK, due to items (f3) and (f4).

Similarly, all unary atoms in the BK are imported by (f1) where the single argument occurs in a binary atom from the BK that has already been imported. Hence, all BK that is relevant for derivations by means of meta-substitutions wrt. forward-chained meta-rules is imported, and the second constraint of item (4) is violated in case a negative example is entailed by $B \cup \mathscr{H}$.

(ii) Every minimal solution $\mathscr{H}$ for $\mathscr{M}$ contains only productive rules as defined right before Theorem 1 in Section 3 because rules which are not productive are not necessary for deriving a positive example. Since we only consider forward-chained meta-rules, an answer set S of $\Pi_f(\mathscr{M})$ such that $\mathscr{H}$ is the logic program induced by S only needs to contain those binary atoms from the BK that occur in a chain that connects the first argument of each positive example $e^+ \in E^+$ to its second argument because only those atoms are necessary for ensuring that each rule in $\mathscr{H}$ is productive. Now, answer sets of $\Pi_f(\mathscr{M})$ are modulo the guess in (3a) least models that can be constructed bottom up incrementally in a fixpoint iteration, and that contain the atoms they logically entail. Hence, all atoms in the corresponding chain are incrementally imported from the BK by the rules in items (f2) and (f4). Accordingly, there is an answer set S of $\Pi_f(\mathscr{M})$ s.t. the induced logic program $\mathscr{H}$ wrt. S is a minimal solution for $\mathscr{M}$.

Note that this suffices for finding minimal solutions in practice as our implementation finds any productive solution that is encoded by $\Pi_f(\mathscr{M})$. $\square$

Proof Sketch for Theorem 3

This result can be shown similarly as the previous Theorem 2. Each unary and binary atom introduced via the items (s1) and (s2) of Definition 12, respectively, whose arguments occur in an acyclic sequence of binary atoms from the BK that connects the first argument a of each positive example $p(a,b) \in E^+$ to its second argument b, can be mapped to exactly one unary and binary atom, resp., that is introduced by items (f1) and (f2) of Definition 7. In this regard, the only difference is that object-level constants in (f1) and (f2) are replaced by abstract states of the form (e^+_{id}, seq_{id}, i) according to Definition 9 in (s1) and (s2). Furthermore, all acyclic sequences representing a possible chain that connects the first argument of each positive example to its second argument are imported by the external atom in item $(s4)$.

Then, the only essential remaining differences between $\Pi_f(\mathscr{M})$ and $\Pi_{sa}(\mathscr{M})$ consist in the fact that sequences that are modeled by a solution and correspond to derivations of positive examples are guessed in item (s4), and that instead of the second constraint from item (4) of Definition 3, the derivability of negative examples is checked by means of the external atom in item (s7). However, a minimal solution for $\mathscr{M}$ needs to model at least one sequence for deriving each positive example, which is ensured jointly by items (s3), (s5) and (s6). Moreover, no minimal solution w.r.t. the restriction to acyclic sequences of Part (ii) of Theorem 3 is lost by excluding cyclic sequences in Definition 8. Finally, item (s7) removes like the second constraint from item (4) all hypotheses that entail a negative example. $\square$

Appendix B Benchmark Encodings and Sample Instance

To illustrate the concrete encodings and the instances employed for the empirical evaluation in Section 5, we present the encodings $\Pi_f(\mathscr{M})$ and $\Pi_{sa}(\mathscr{M})$ as well as the input to Metagol used for the *Robot Waiter Strategies* benchmark (B3). A sample instance and a corresponding solution of benchmark (B3) can be found at the end of this section.

Moreover, the encodings of all benchmark problems used in Section 5 and all instances used in the experiments are available at http://www.kr.tuwien.ac.at/research/projects/inthex/hexmil/.

B.1 Forward-Chained HEX*-MIL-Encoding*

```
binary(pour_tea,X,Y) :- &pour_tea[X](Y), state(X).
binary(pour_coffee,X,Y) :- &pour_coffee[X](Y), state(X).
binary(move_right,X,Y) :- &move_right[X](Y), state(X).

unary(wants_tea,X) :- &wants_tea[X](), state(X).
unary(wants_coffee,X) :- &wants_coffee[X](), state(X).
unary(at_end,X) :- &at_end[X](), state(X).

order(X,Y) :- skolem(X), binary(Y,_,_).
order(X,Y) :- pos_ex(X,_,_), binary(Y,_,_).
order(X,Y) :- pos_ex(X,_,_), skolem(Y).
order(X,Y) :- skolem(X), skolem(Y), X < Y.

{meta(precon,P1,P2,P3)} :- order(P1,P3), unary(P2,X), deduced(P3,X,Y).
{meta(postcon,P1,P2,P3)} :- order(P1,P2), deduced(P2,X,Y), unary(P3,Y).
{meta(chain,P1,P2,P3)} :- order(P1,P2), order(P1,P3), deduced(P2,X,Z),
                                                          deduced(P3,Z,Y).
{meta(tailrec,P1,P2,n)} :- order(P1,P2), deduced(P2,X,Z), deduced(P1,Z,Y).

deduced(P1,X,Y) :- meta(precon,P1,P2,P3), unary(P2,X), deduced(P3,X,Y).
deduced(P1,X,Y) :- meta(postcon,P1,P2,P3), deduced(P2,X,Y), unary(P3,Y).
deduced(P1,X,Y) :- meta(chain,P1,P2,P3), deduced(P2,X,Z), deduced(P3,Z,Y).
deduced(P1,X,Y) :- meta(tailrec,P1,P2,n), deduced(P2,X,Z), deduced(P1,Z,Y).

state(X) :- pos_ex(_,X,_).
state(Y) :- deduced(_,_,Y).

deduced(P,X,Y) :- binary(P,X,Y).

:- pos_ex(P,X,Y), not deduced(P,X,Y).
:- pos_ex(P,X,Y1), deduced(P,X,Y2), Y1 != Y2.

:- #count{ M,P1,P2,P3 : meta(M,P1,P2,P3) } != N, size(N).
```

B.2 State Abstraction HEX*-MIL-Encoding*

Note that even though the syntax of the external atom used for importing binary and unary BK as well as the positive examples in the encoding below differs from the external atoms used in Definition 12, identical extensions are imported for the atoms *binary*, *unary* and *pos* as described in Section 4. Hence, the encoding is equivalent to the encoding of Definition 12.

```
binary(A,N1,N2) :- &abduceSequence[ID,ExStart,ExEnd](X,N1,N2,A),
                             pos_ex(ID,_,ExStart,ExEnd), X = seq.
unary(A,N) :- &abduceSequence[ID,ExStart,ExEnd](X,N,N,A),
                             pos_ex(ID,_,ExStart,ExEnd), X = check.
```

```
pos(ID,A,N1,N2) v n_pos(ID,A,N1,N2) :-
                              &abduceSequence[ID,ExStart,ExEnd](X,N1,N2,A),
                              pos_ex(ID,_,ExStart,ExEnd), X = goal.

:- &failNeg[meta,pos_ex]().

pos1(ID) :- pos(ID,_,_,_).
:- pos_ex(ID,_,_,_), not pos1(ID).

order(X,Y) :- skolem(X), binary(Y,_,_).
order(X,Y) :- pos(X,_,_), binary(Y,_,_).
order(X,Y) :- pos(X,_,_), skolem(Y).
order(X,Y) :- skolem(X), skolem(Y), X < Y.

meta(precon,P1,P2,P3) :- order(P1,P3), unary(P2,X), deduced(P3,X,Y).
meta(postcon,P1,P2,P3) :- order(P1,P2), deduced(P2,X,Y), unary(P3,Y).
meta(chain,P1,P2,P3) :- order(P1,P2), order(P1,P3), deduced(P2,X,Z),
                                                      deduced(P3,Z,Y).
meta(tailrec,P1,P2,n) :- order(P1,P2), deduced(P2,X,Z), deduced(P1,Z,Y).

deduced(P1,X,Y) :- meta(precon,P1,P2,P3), unary(P2,X), deduced(P3,X,Y).
deduced(P1,X,Y) :- meta(postcon,P1,P2,P3), deduced(P2,X,Y), unary(P3,Y).
deduced(P1,X,Y) :- meta(chain,P1,P2,P3), deduced(P2,X,Z), deduced(P3,Z,Y).
deduced(P1,X,Y) :- meta(tailrec,P1,P2,n), deduced(P2,X,Z), deduced(P1,Z,Y).

deduced(P,X,Y) :- binary(P,X,Y).

:- not deduced(P,X,Y), pos(_,P,X,Y).

:- #count M,P1,P2,P3 : meta(M,P1,P2,P3)  != N, size(N).
```

B.3 Metagol Input Program

```
metagol:functional.

func_test(Atom,PS,G):-
  Atom = [P,A,B],
  Actual = [P,A,Z],
\+ (metagol:prove_deduce([Actual],PS,G),Z \= B).

metarule(precon,[P,Q,R],([P,A,B]:-[[Q,A],[R,A,B]])).
metarule(postcon,[P,Q,R],([P,A,B]:-[[Q,A,B],[R,B]])).
metarule(chain,[P,Q,R],([P,A,B]:-[[Q,A,C],[R,C,B]])).
metarule(tailrec,[P,Q],([P,A,B]:-[[Q,A,C],[P,C,B]])).

prim(pour_tea/2).
prim(pour_coffee/2).
prim(move_right/2).

prim(wants_tea/1).
prim(wants_coffee/1).
prim(at_end/1).

a :-
```

```
  train_exs(Pos),
  Neg = [],
  learn(Pos,Neg).


pour_tea([robot_pos(X),end(Y),places([place(X,A,cup(up,empty))|R])],
                    [robot_pos(X),end(Y),places([place(X,A,cup(up,tea))|R])]).
pour_tea([robot_pos(X),end(Y),places([E|R1])],
                                     [robot_pos(X),end(Y),places([E|R2])]) :-
pour_tea([robot_pos(X),end(Y),places(R1)],[robot_pos(X),end(Y),places(R2)]).

pour_coffee([robot_pos(X),end(Y),places([place(X,A,cup(up,empty))|R])],
                 [robot_pos(X),end(Y),places([place(X,A,cup(up,coffee))|R])]).
pour_coffee([robot_pos(X),end(Y),places([E|R1])],
                                     [robot_pos(X),end(Y),places([E|R2])]) :-
pour_coffee([robot_pos(X),end(Y),places(R1)],[robot_pos(X),end(Y),places(R2)]).

move_right([robot_pos(X1),end(Y)|R],[robot_pos(X2),end(Y)|R]) :-
                                                   X1 < Y, X2 is X1 + 1.

wants_tea([robot_pos(X),end(_),places([place(X,tea,_)|_])]).
wants_tea([robot_pos(X),end(Y),places([_|R])]) :-
                                  wants_tea([robot_pos(X),end(Y),places(R)]).

wants_coffee([robot_pos(X),end(_),places([place(X,coffee,_)|_])]).
wants_coffee([robot_pos(X),end(Y),places([_|R])]) :-
                               wants_coffee([robot_pos(X),end(Y),places(R)]).

at_end([robot_pos(X),end(X)|_]).
```

B.4 Sample Instance and Solution

```
Instance:

pos_ex([robot_pos(1),end(3),places([place(1,coffee,cup(up,empty)),
place(2,coffee,cup(up,empty))])],
[robot_pos(3),end(3),places([place(1,coffee,cup(up,coffee)),
place(2,coffee,cup(up,coffee))])]).

pos_ex([robot_pos(1),end(6),places([place(1,coffee,cup(up,empty)),
place(2,coffee,cup(up,empty)),place(3,coffee,cup(up,empty)),
place(4,tea,cup(up,empty)),place(5,coffee,cup(up,empty))])],
[robot_pos(6),end(6),places([place(1,coffee,cup(up,coffee)),
place(2,coffee,cup(up,coffee)),place(3,coffee,cup(up,coffee)),
place(4,tea,cup(up,tea)),place(5,coffee,cup(up,coffee))])]).

Solution:

robot(A,B):-robot_1(A,B),at_end(B).
robot(A,B):-robot_1(A,C),robot(C,B).
robot_1(A,B):-robot_2(A,C),move_right(C,B).
robot_2(A,B):-wants_tea(A),pour_tea(A,B).
robot_2(A,B):-wants_coffee(A),pour_coffee(A,B).
```