Typed meta-interpretive learning of logic programs

Rolf Morel, Andrew Cropper, and Luke Ong

University of Oxford

JELIA 2019

Setting the scene

- Inductive Logic Programming

```
droplasts(["jelia","2019"],["jeli","201"]).
droplasts(["rende","cyprus","madeira"],
          ["rend","cypru","madeir"]).
```

- Meta-Interpretive Learning (MIL) framework
  - Provide BK predicates such as *map*/3, *reverse*/2, and *tail*/2,
  - Provide metarules such as $P(A, B) \leftarrow Q(A, C), R(C, B)$
- Logic programs are typically untyped

```
droplasts(A,B) :- map(A,C,droplasts_1).
droplasts_1(A,B) :- reverse(A,C),droplasts_2(C,B).
droplasts_2(A,B) :- tail(A,C),reverse(C,B).
```

Contributions

Prune hypothesis search space by type checking

- Extend MIL to support polymorphic types
- Hypothesis space reduction by a cubic factor
- Inference of polymorphic types for invented predicates
- Implementations in `Prolog` and `ASP`
- Experimental reduction in learning times

Terms and types

Notation ":" means "has type"

Ground terms have types:

- $a : char$
- $[1, 2] : list(int)$

Non-ground terms have types:

- $[H|T] : list(S)$
- $A : T$

Atoms/predicates have types:

- $succ(A, B) : (int, int)$
- $P(A, B) : (U, V)$
- $head([H|\_], H) : (list(T), T)$
- $map(A, B, F) : (list(U), list(V), (U, V))$

Typed (meta-)rules and typed logic programs

Chain rule

- $P(A, B) \leftarrow Q(A, C), R(C, B)$

  becomes
- $P(A, B){:}(Ta, Tb) \leftarrow Q(A, C){:}(Ta, Tc), R(C, B){:}(Tc, Tb)$

A metarule (resp. a logic program) is typed if all atoms are typed.

The ":" notation is sugar:

- $P(A_1, \ldots, A_n) : (\tau_1, \ldots, \tau_n)$

  can be represented as
- $P(\tau_1, \ldots, \tau_n, A_1, \ldots, A_n)$

Type definition and type terms

Variables, constant and function symbols:

- Set of type variables V_t
- $T_b \subseteq \mathcal{C}$ of base types (e.g. int)
- $T_c \subseteq \mathcal{F}$ of polymorphic type constructors (e.g. $list/1$)

The set $\mathcal{T}$ of types has members such as:

- Data types: $bool$, $list(int)$, $record(int, list(T))$
- Predicate types: (int, int), $(list(T), T)$
- Higher-order polymorhic types: $(list(S), list(T), (S, T))$

Typed MIL problem

Typed MIL input:

- $BK = B_C \cup M$
  - B_C is a set of **typed** Horn clauses
  - M is a set of **typed** metarules
- Examples E^+ and E^- are **typed** ground atoms

Typed MIL problem:

Find a **typed** logic program hypothesis H such that

- $H \cup B_C \models E^+$
- $H \cup B_C \not\models E^-$

Hypothesis space reduction

(Def) **Type relevancy**
A predicate symbol is *type relevant* if it there exists a hypothesis that is type consistent with the BK and the examples.

(Example) Given BK:

```
map(A,B,F):(list(U),list(V),(U,V))
reverse(A,B):(list(T),list(T))
tail(A,B):(list(T),list(T))
succ(A,B):(int,int)
```

Then there is no type consistent hypothesis that uses *succ*/2 for:

```
droplasts(["jelia","2019"],["jeli","201"]):
            (list(list(char)),list(list(char))).
```

Hypothesis space reduction (in $\mathcal{H}_2^2$)

For simplicity consider the hypothesis space $\mathcal{H}_2^2$

- At most 2 literals in clause bodies (arities ≤ 2)
- Hypothesis space size $\leq (mp^3)^n$
  - m is #metarules, p is #predicate symbols, n is #clauses

(Def) **Relevant ratio**
Given p' type relevant predicate symbols, the relevant ratio is $r = p'/p$.

(Thm) **Hypothesis space reduction**
Given p predicate symbols, and a relevant ratio r, typing reduces the MIL hypothesis space by a factor of r^{3n}.

Replace p with rp above to obtain size $\leq r^{3n}(mp^3)^n$. $\square$

Implementations

Prolog implementation Metagol$_T$:

- Supports higher-order predicates and inventions
- Atoms annotated with *derivation types*
  - e.g. $P([1,2,3],3)$:$(list(int), int)$
  - Types that are accurate for argument values
- Atoms additionally annotated with *general types*
  - e.g. $P([1,2,3],3)$:$(list(int), int)$:$(list(T), int)$
  - Types that are accurate for how a predicate may be used
  - *least general generalizations* of derivation types
- Type checking is just unification
  - $head(A, B)$:$(list(T), T)$ would be tried for P, but $tail(A, B)$:$(list(T), list(T))$ would not.

Implementations

ASP implementation HEXMIL_T:

- Based on HEXMIL, an Answer Set Programming MIL encoding
- Each atom is given additional arguments to represent the types
  - E.g., was `binary_bg(succ,A,B):-B=A+1,state(A).`
  - `binary_bg(succ,(int,int),A,B):-B=A+1,state(A,int).`
- Extension of HEXMIL encoding for higher-order predicates

Experiment: `droplasts/2`

Examples:

```
droplasts(["jelia","2019"],["jeli","201"]).
droplasts(["rende","cyprus","madeira"],
          ["rend","cypru","madeir"]).
```

Target program:

```
droplasts(A,B) :- map(A,C,droplasts_1).
droplasts_1(A,B) :- reverse(A,C),droplasts_2(C,B).
droplasts_2(A,B) :- tail(A,C),reverse(C,B).
```

Experiment: droplasts/2

Target program with types:

```
droplasts(A,B):(list(list(T)),list(list(T))):-
    map(A,C,droplasts_1)
      :(list(list(T)),list(list(T)),(list(T),list(T))).
droplasts_1(A,B):(list(T),list(T)):-
    reverse(A,C):(list(T),list(T)),
    droplasts_2(C,B):(list(T),list(T)).
droplasts_2(A,B):(list(T),list(T)):-
    tail(A,C):(list(T),list(T)),
    reverse(C,B):(list(T),list(T)).
```

Experiment: `droplasts/2`

Sample *small* examples at random.

Sample BK predicates from:

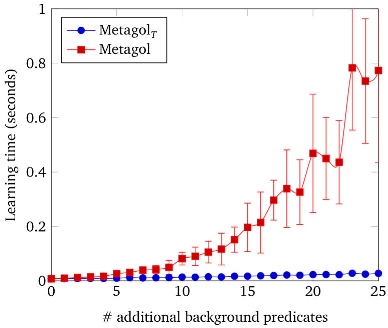
tail(A,B):(list(T),list(T)).	succ(A,B):(int,int).
map(A,B,F):(list(T),list(S),(S,T)).	last(A,B):(list(T),T).
reverse(A,B):(list(T),list(T)).	min_list(A,B):(list(int),int).
sumlist(A,B):(list(int),int).	pred(A,B):(int,int).
head(A,B):(list(T),T).	max_list(A,B):(list(int),int).

Experiment: **vary the number of background predicates**

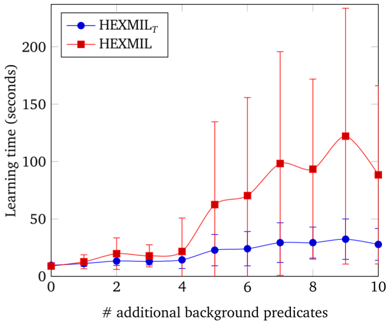
Always include *map*/3, *reverse*/2, and *tail*/2

Experiment: `droplasts/2`

Results



Prolog



ASP

Future work

- Decidability proof:
  - Types involve functional symbols
  - Still can argue for finite number of types
  - First-order is clear, higher-order is not
- More complex types:
  - Union types
  - Refinement types (types restricted by propositions):
    - Attempted with SMT solving
    - Pure prolog shows some advantage
- Type invention