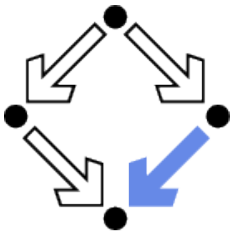
Learning Higher-Order Logic Programs From Failures

Stanisław J. Purgał, **David M. Cerna**,and Cezary Kaliszyk

July 23^{rd}-29^{th} 2022
IJCAI-22



Inductive Logic Programming (ILP)

- is a form of **symbolic machine learning** *(Muggleton, 1991)*.
- Learning From Entailment: Find a logic program **H** s.t.

$$BK, \mathbf{H} \models E^+ \qquad BK, \mathbf{H} \not\models E^-$$

BK	E^+	E^-
$\mathtt{mom}(a,b).\ \mathtt{dad}(e,b).$	$\mathtt{gp}(a,d).\ \mathtt{gp}(e,d).$	$\mathtt{gp}(a,b).\ \mathtt{gp}(b,c).$
$\mathtt{mom}(a,c).\ \mathtt{dad}(e,c).$	$\mathtt{gp}(a,f).\ \mathtt{gp}(e,f).$	$\mathtt{gp}(c,f).\ \mathtt{gp}(d,f).$
$\mathtt{mom}(b,d).\ \mathtt{dad}(c,f).$		

Solutions: Predicate Invention (PI) Improves Generalization

$\mathtt{gp}(X,Y)\text{:-}\mathtt{mom}(X,C), \mathtt{mom}(C,Y)$
$\mathtt{gp}(X,Y)\text{:-}\mathtt{mom}(X,C), \mathtt{dad}(C,Y)$
$\mathtt{gp}(X,Y)\text{:-}\mathtt{dad}(X,C), \mathtt{mom}(C,Y)$
~~$\mathtt{gp}(X,Y)\text{:-}\mathtt{dad}(X,C), \mathtt{dad}(C,Y)$~~

$\mathtt{gp}(X,Y)\text{:-}\mathbf{p}(X,C), \mathbf{p}(C,Y)$
$\mathbf{p}(X,Y)\text{:-}\mathtt{mom}(X,Y)$
$\mathbf{p}(X,Y)\text{:-}\mathtt{dad}(X,Y)$

Concise Solutions Through Higher-Order (HO) Definitions

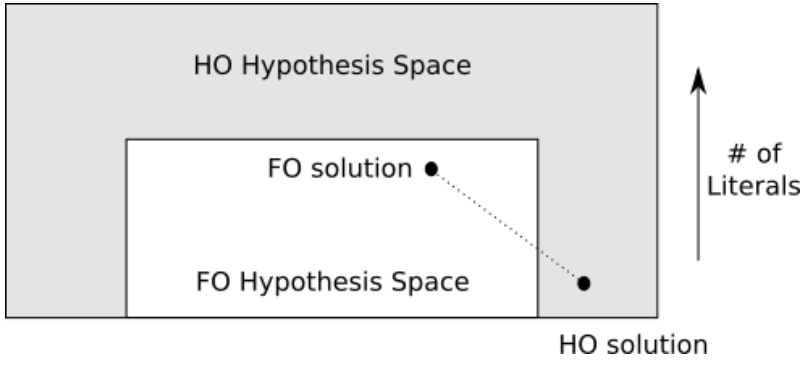
First-Order

```
ASN(A, B):- zero(C), h(C, A, D), g(D, B).
  g(A, B):- empty(A), empty(B).
  g(A, B):- head(A, C), tail(A, F), zero(E), h(E, C, D),
            g(F, G), tail(B, G), head(B, D).
h(A, B, C):- zero(B), empty(C).
h(A, B, C):- pred(B, D), suc(A, E), less0(B),
            h(E, D, F), tail(C, F), head(C, E).
```

Higher-Order

```
ASN(A, B):- zero(C), ite(p, C, A, D),
            map(q, D, B).
  p(A, B):- suc(A, B).
  q(A, B):- zero(C), ite(p, C, A, B).
```

- Improves generalization and conciseness of solutions.
- Existing approaches use templates and limit usage of definitions.
- **Hopper** exploits *Learning from failures* (Cropper & Morel, 2021) to learn large and complex programs using Higher-order definitions.



Meta-Interpretive Learning (MiL): a Step Towards HO

- Integration of HO reasoning into ILP can be traced to the early days *(Feng and Muggleton, 1992)*.
- Effective use intimately connected to **PI**.

$$\texttt{reverse}(A, B) \texttt{:-}\ \texttt{empty}(C),\ \texttt{fold}(\textbf{???}, C, A, B).$$

Meta-Interpretive Learning (MiL): a Step Towards HO

- Integration of HO reasoning into ILP can be traced to the early days *(Feng and Muggleton, 1992)*.
- Effective use intimately connected to **PI**.

$$\texttt{reverse}(A, B) \texttt{ :- } \texttt{empty}(C),\ \texttt{fold}(\mathbf{p}, C, A, B).$$

$$\texttt{p}(A, B, C)\texttt{:- } \texttt{head}(C, B),\ \texttt{tail}(C, A).$$

- Then came **MiL** *(Muggleton et al. 2014)*:
  - Constraint the search space using second-order horn clauses.
  - Instantiate predicate variables using **BK** or,
  - Invented predicates defined using the Metarules:

$$P(X, Y) \text{ :- } Q(X, Z), R(Z, Y) \quad \text{(Chain Rule)}$$

$$P(X, Y) \text{ :- } Q(X, Z), P(Z, Y) \quad \text{(Dyactic Recursion)}$$

HO Definitions as Interpreted BK

$$\texttt{map}(P, X, Y) \texttt{:-}\ \texttt{empty}(X), \texttt{empty}(Y).$$
$$\texttt{map}(P, L_1, L_2) \texttt{:-}\ \texttt{cons}(L_1, H_1, T_1),\ \texttt{cons}(L_2, H_2, T_2),$$
$$\texttt{P}(H_1, H_2),\ \texttt{map}(P, T_1, T_2).$$

- HO Definitions as a type of metarule (Cropper et al. 2020):
  - partially-instantiated second-order variables.
  - second-order arguments.
- **Metagol**$_{ho}$ successfully finds HO programs.
- Problems arise with complex referencing of definitions:

$$\texttt{half}(A, B) \texttt{:-}\ \texttt{reverse}(A, C), \texttt{caseList}(p, q, r, C, B).$$
$$\texttt{p}(A) \texttt{:-}\ \texttt{empty}(A).$$
$$\texttt{q}(A, B) \texttt{:-}\ \texttt{empty}(B).$$
$$\texttt{r}(A, B, C) \texttt{:-}\ \texttt{front}(B, E), \texttt{caseList}(p, q, r, E, D), \texttt{app}(D, A, C).$$

- Implies instantiation of a variable by a metarule?!

Popper: Learning from Failures (LFF)

- LFF (Cropper & Morel, 2021) is an ILP paradigm which:
  - **Generates** a plausible hypotheses H.
  - **Test** H against E^+ and E^-.
  - **Constrains** the generator based on the tester's results.
  - **Repeat** till task is solved.
- Constraints are based on Subsumption:
  - **Generalization**: eliminate all H' that subsume H.
  - **Specialization**: eliminate all H' subsumed by H.
- Soundness follows from $H \leq H' \Rightarrow H \models H'$.
  - Does not necessarily hold when H contains HO definitions.

$$\{h(X,Y)\texttt{:-map}(???,Y,Z),\cdots\} \leq_{sub} \{h(X,Y)\texttt{:-map}(???,X,Z),\cdots\}$$

- Comparision requires knowing something about ???.

Generate Principal Programs Instead!

- Every **Instance** of a HO Definition is associated with a **Unique** invented predicate.
- No HO arguments needed.

$$\texttt{reverse}(A, B) \texttt{:-}\ \texttt{empty}(C),\ \texttt{fold}_a(C, A, B).$$
$$\texttt{fold}_{p\_a}(A, B, C) \texttt{:-}\ \texttt{head}(C, B),\ \texttt{tail}(C, A).$$

Generate Principal Programs Instead!

- Every **Instance** of a HO Definition is associated with a **Unique** invented predicate.
- No HO arguments needed.

$$\texttt{reverse}(A, B) \texttt{:-}\ \texttt{empty}(C),\ \texttt{fold}_a(C, A, B).$$
$$\texttt{fold}_{p\_a}(A, B, C) \texttt{:-}\ \texttt{head}(C, B),\ \texttt{tail}(C, A).$$
$$\texttt{fold}_a(A, B, C) \texttt{:-}\ \texttt{fold}(\texttt{fold}_{p\_a}, A, B, C).$$

Generate Principal Programs Instead!

- Every **Instance** of a HO Definition is associated with a **Unique** invented predicate.
- No HO arguments needed.

$$\begin{aligned}
\texttt{reverse}(A,B) &\texttt{:-}\ \texttt{empty}(C),\ \texttt{fold}_a(C,A,B).\\
\texttt{fold}_{p_a}(A,B,C) &\texttt{:-}\ \texttt{head}(C,B),\ \texttt{tail}(C,A).\\
\texttt{fold}_a(A,B,C) &\texttt{:-}\ \texttt{fold}(\texttt{fold}_{p_a},A,B,C).\\
\texttt{fold}(P,A,B,C) &\texttt{:-}\ \texttt{empty}(B), A = C.\\
\texttt{fold}(P,A,B,C) &\texttt{:-}\ \texttt{head}(B,H), P(H,D),\\
&\qquad\qquad \texttt{tail}(B,T), \texttt{fold}(P,D,T,C).
\end{aligned}$$

- The **generator** produces first-order programs that may encode instances of HO definitions.
- Introduces many incomparable programs which are equivalent.
- **But**, it is effective!

Experimental Results

Task	*Popper* **(Opt)**	#Literals	PI?	*Hopper*	**Hopper (Opt)**	#Literals	HO-Predicates	*Metagol*$_{\mathrm{HO}}$	Metatypes?
Learning Programs by learning from Failures (Cropper *et al.*, 2021)									
dropK	1.1s	7	no	0.5s	**0.1s**	4	iterate	no	no
allEven	0.2s	7	no	0.2s	**0.1s**	4	all	yes	no
findDup	0.25s	7	no	−−	**0.5s**	10	caseList	no	yes
length	0.1s	7	no	0.2s	**0.1s**	5	fold	yes	no
member	0.1s	5	no	0.2s	**0.1s**	4	any	yes	no
sorted	65.0s	9	no	46.3s	**0.4s**	6	fold	yes	no
reverse	11.2s	8	no	7.7s	**0.5s**	6	fold	yes	no
Learning Higher-Order Logic Programs (Cropper *et al.*, 2020)									
dropLast	300.0s	10	no	300s	**2.9s**	6	map	yes	no
encryption	300.0s	12	no	300s	**1.2s**	7	map	yes	no
Additional Tasks									
repeatN	5.0s	7	no	0.6s	**0.1s**	5	iterate	yes	no
rotateN	300.0s	10	no	300s	**2.6s**	6	iterate	yes	no
allSeqN	300.0s	25	yes	300s	**5.0s**	9	iterate, map	yes	no
dropLastK	300.0s	17	yes	300s	**37.7s**	11	map	no	no
firstHalf	300.0s	14	yes	300s	**0.2s**	9	iterateStep	yes	no
lastHalf	300.0s	12	no	300s	**155.2s**	12	caseList	no	yes
of1And2	300.0s	13	no	300s	**6.9s**	13	try	no	no
isPalindrome	300.0s	11	no	157s	**2.4s**	9	condlist	no	yes
depth	300.0s	14	yes	300s	**10.1s**	8	fold	yes	yes
isBranch	300.0s	17	yes	300s	**25.9s**	12	caseTree, any	no	yes
isSubTree	2.9s	11	yes	1.0s	**0.9s**	7	any	yes	yes
addN	300.0s	15	yes	300s	**1.4s**	9	map, caseInt	yes	no
mulFromSuc	300.0s	19	yes	300s	**1.2s**	7	iterate	yes	no

- **Hopper** is significantly faster than **Popper**, and
- solves more task than both **Popper** and **Metagol**$_{HO}$.

Acknowledgements

Supported by the ERC starting grant no. 714034 SMART, the MathLP project (LIT- 2019-7-YOU-213) of the Linz Institute of Technology and the state of Upper Austria, Cost action CA20111 EuroProofNet, and project CZ.02.2.69/0.0/0.0/18 053/0017594 of the Ministry of Education, Youth and Sports of the Czech Republic.

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education