Hopper: Learning Higher-Order Logic Programs from Failures

Seminar in Knowledge Representation and Reasoning – 184.712 – 2023S

Simão Costa (e12202234)

Inductive Logic Programming (ILP)

• form of symbolic machine learning that aims to learn a logic program from **background knowledge (BK)** predicates / examples

• uses **First-order (FO) logic** to represent hypotheses and data

• the output of an ILP system is a logic program that can be used to make predictions or perform reasoning in the given domain

Inductive Logic Programming (ILP)

- **excessively large BK** can, in many cases, lead to **performance loss**

    - increases the search space and computational complexity

    - higher chance of irrelevant information being present, introducing unnecessary complexity / misleading the ILP system

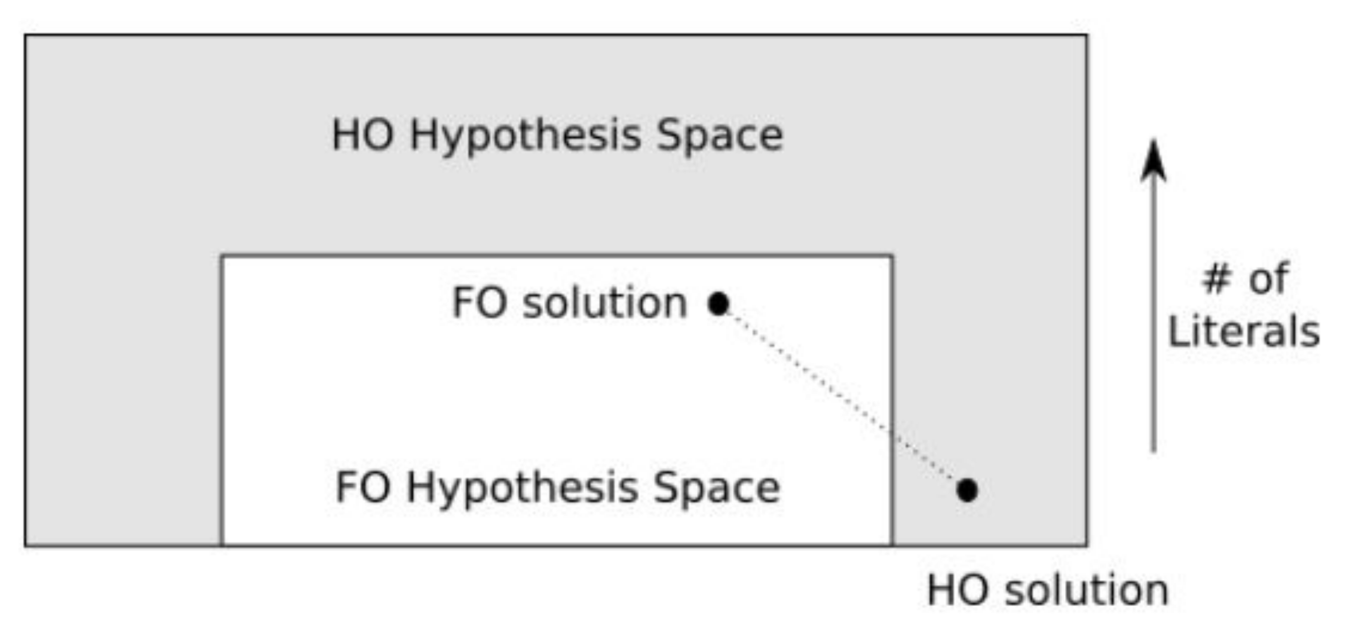
Large BK brings some problems!

High-order (HO) definitions

- offer several **benefits** compared to background knowledge
    - provide increased **expressiveness**
    - facilitate generalization and abstraction

Allow ILP systems to capture higher-level patterns and handle new and unseen instances, enabling a more **efficient learning** and **reasoning**

High-order (HO) definitions



Predicate Invention (PI)

- The effective use of HO predicates is closely tied to **Predicate Invention (PI)**
  - allows for the creation of new predicates

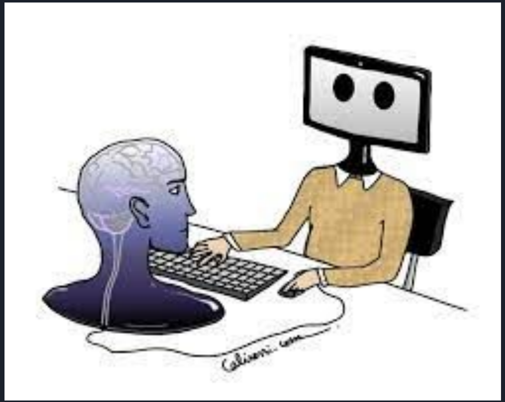
$$\boxed{\begin{array}{c}\texttt{reverse}(A,B)\text{:- }\texttt{empty}(C),\ \texttt{fold}(p,C,A,B).\\ \texttt{p}(A,B,C)\text{:- }\texttt{head}(C,B),\ \texttt{tail}(C,A).\end{array}}$$

Many well-known ILP frameworks (*Foil, Progol, Tilde, Aleph*) **do not support** predicate invention

HO-enabled ILP

- existing HO-enabled ILP systems are based on **Meta-interpretive Learning (MiL)**
    - **efficiency** and **performance** of MiL-based systems is dependent on human guidance in the form of **metarules**

rules or templates that guide the search process for learning logic programs



Metarules

- define the **higher-order relationships** and **patterns** that the ILP system should consider during the learning process

- guide the ILP system towards specific higher-order relationships or structures that are relevant to the problem domain

- Metarules act as constraints!

Definition 1 ([Cropper and Tourret, 2020]) *A* metarule *is a second-order Horn clause of the form* $A_0 \leftarrow A_1, \cdots, A_n$*, where* A_i *is a literal* $P(T_1, \cdots, T_m)$*, s.t.* P *is either a predicate symbol or a HO variable and each* T_i *is either a constant or a FO variable.*

Metarules

- In complex ILP tasks there may be a multitude of potential metarules that could be relevant to the problem domain
    - selecting the most suitable metarules involves understanding patterns, relationships, and dependencies in the data

The expertise of a knowledgeable human expert is often necessary

Example

- iterating over a sequence of numbers and finding the sum of all the elements
- "iterate" predicate with three arguments: the current element, the next element, and the running total
    - updating the running total by adding the current element and passing it on as the total for the next iteration

HEXMILHO - only supports **binary** definitions - directly representing the "iterate" concept becomes challenging

- “iterate” is **ternary**

Limit human involvement!

Higher-order Metagol

- The HO Metagol is a MiL algorithm implemented using a Prolog meta-interpreter
- It takes as input: **predicate declarations**, sets of **positive and negative examples**, **compiled background knowledge** and a set of **metarules**
    - Invented predicates are introduced if the background knowledge is insufficient

- MetagolHO extends Metagol by including **interpreted background knowledge (BKin)**
    - allows for the handling of additional predicates in a similar way to metarules

Higher-order Metagol

$$
\begin{aligned}
\mathtt{half}_{\mathtt{1st}}(A,B)\text{:- } & \mathtt{reverse}(A,C), \\
& \mathtt{case}_{\mathtt{list}}(p_{[\,]}, p_{[H|T]}, C, B). \\
\mathtt{p}_{[\,]}(A)\text{:- } & \mathtt{empty}(A). \\
\mathtt{p}_{[H|T]}(A,B,C)\text{:- } & \mathtt{empty}(B), \mathtt{empty}(C). \\
\mathtt{p}_{[H|T]}(A,B,C)\text{:- } & \mathtt{front}(B,D)^4, \\
& \mathtt{case}_{\mathtt{list}}(p_{[\,]}, p_{[H|T]}, D, E), \\
& \mathtt{append}(E,A,C).
\end{aligned}
$$

Higher-order HEXMIL

- HEXMIL is an ASP encoding of MiL which uses the HEX formalism to interface with external resources
    - HEXMIL is restricted to forward-chained metarules

Definition 2 Forward-chained *metarules are of the form:* $P(A,B) \text{ :- } Q_1(A,C_1), Q_2(C_1,C_2), \cdots, Q_n(C_{n-1},B), R_1(D_1), \cdots, R_m(D_m)$ *where* $D_i \in \{A, C_1, \cdots, C_{n-1}, B\}$.

- only **Dyadic** learning task may be handled

Popper: Learning From Failures (LFF)

- based on counterexample guided inductive synthesis (CEGIS)
- introduced in [Cropper and Morel, 2021a]

- predicate declarations (PD)
- sets of positive (E+) and negative (E-) examples
- background knowledge (BK)

Inputs

Popper: Learning From Failures (LFF)

- **learning process** -> generate-test-constrain loop

Generate phase

- candidate programs are chosen from the hypothesis space

possible programs that have not been ruled out yet

- tested against the positive and negative examples in the **test phase**

Popper: Learning From Failures (LFF)

Test phase

- the program is tested against E^+ and E^-
- If a candidate program correctly entails all positive examples and none of the negative examples - Popper **terminates** and considers it a **successful solution**

Constrain phase

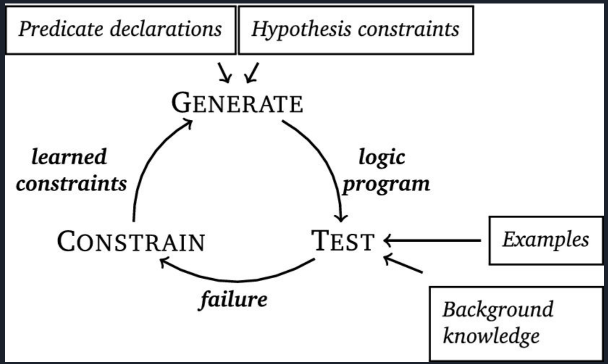
- introduce constraints - based on **negative examples**
- The choice of constraints is guided by **Θ-subsumption**

Popper: Learning From Failures (LFF)

- Popper iterates through the loop, gradually refining the hypothesis space
- It aims to find the optimal solution - the program containing the fewest literals

Popper utilizes a multi-shot solving framework and encodes both definite logic programs and constraints within the **Answer Set Programming (ASP)** paradigm

  - incorporating the language bias and other constraints.



Logic Programming Overview

- P -> countable set of predicate symbols ($p, q, r, p_1, \ldots$)
- V_f -> countable set of FO variables ($A, B, C, \ldots$)
- V_h -> countable set of HO variables ($P, Q, R, \ldots$)
- T -> set of FO terms constructed from function symbols and V_f ($s, t, s_1, t_1, \ldots$)
- a -> atom of the form $p(T_1, T_m, t_1, t_n, \ldots)$
- $s_y(a)$ = p -> symbol of the atom
- $ag_h(a) = \{T_1, \ldots, T_m\}$ -> HO-arguments
- $ag_f(a) = \{t_1, \ldots, t_n\}$ -> FO-arguments
  - $ag_h(a) = \varnothing$ and $s_y(a) \in$ P -> a as FO
  - $ag_h(a) \subset P$ and $s_y(a) \in P$ -> a as HO-ground
  - otherwise a as HO
- A literal is either an atom or its negation
- A literal is HO if the atom it contains is HO

Logic Programming Overview

- Clause - set of literals:
  - Horn clauses - **at most one** positive literal
  - Definite clauses - **exactly one** positive literal
- Clause **positive** literal -> **head - hd(c)**
- Clause **negated** literals -> **body - bd(c)**
- Theory -> a finite set of clauses

  ↳ A theory is considered FO if all atoms are FO

- Substitution -> Replacing variables P_1, P_n, A_1, A_m, … by predicate symbols p_1, p_n, … and terms t_1, t_m, … -> (θ, σ, …)
- A substitution θ unifies two atoms when $a\theta = b\theta$.

Hypothesis Space

- **Interpretable** theories -> principal programs - FO clausal theories encoding the relationship literals - clauses, along with HO definitions

Hopper - generates and tests these principal programs

- The encoding of principal programs ensures the validity of the pruning mechanism proposed in past work
    - each principal program represents a unique HO program

Definition 3 *A clause c is* proper[7] *if $ag_h(hd(c))$ are pairwise distinct, $ag_h(hd(c)) \subset \mathcal{V}_h$, and $\forall a \in bd(c)$,*

a) if $sy(a) \in \mathcal{V}_h$, then $sy(a) \in ag_h(hd(c))$, and

b) if $p \in ag_h(a)$ and $p \in \mathcal{V}_h$, then $p \in ag_h(hd(c))$.

A finite set of proper clauses d with the same head (denoted $hd(d)$) is referred to as a *HO definition*. A set of distinct HO definitions is a *library*. Let $\mathcal{P}_{PI} \subset \mathcal{P}$ be a set of predicate symbols reserved for invented predicates.

- a -> atom of the form $p(T_1, T_m, t_1, t_n, \ldots)$
- $s_y(a)$ = p -> symbol of the atom
- $ag_h(a) = \{T_1, \ldots, T_m\}$ -> HO-arguments
- $ag_f(a) = \{t_1, \ldots, t_n\}$ -> FO-arguments
- P -> countable set of predicate symbols (*p, q, r,* p_1, ...)
- Clause **positive** literal -> **head - hd(c)**
- Clause **negated** literals -> **body - bd(c)**
- V_f -> countable set of FO variables (*A, B, C,* ...)
- V_h -> countable set of HO variables (*P, Q, R,* ...)

Definition 4 *A f.f.d theory* $\mathfrak{T}$ *is* interpretable *if* $\forall c \in \mathfrak{T}$, $ag_h(hd(c)) = \emptyset$ *and* $\forall l \in bd(c)$, *l is higher-order ground,*

a) *if* $ag_h(l) \neq \emptyset$, *then* $\forall c' \in \mathfrak{T}$, $sy(hd(c')) \neq sy(l)$, *and*

b) $\forall p \in ag_h(l), \exists c' \in \mathfrak{T}$, *s.t.* $sy(hd(c')) = p \in \mathcal{P}_{PI}$.

Atoms s.t. $ag_h(l) \neq \emptyset$ *are* external. *The set of external atoms of an interpretable theory* $\mathfrak{T}$ *is denoted by* $ex(\mathfrak{T})$.

- a -> atom of the form *p*(T_1, T_m, t_1, t_n, ...)
- s_y(a) = p -> symbol of the atom
- ag_h(a) = {T_1, ..., T_m} -> HO-arguments
- ag_f(a) = {t_1, ..., t_n} -> FO-arguments
- P -> countable set of predicate symbols (*p*, *q*, *r*, p_1, ...)
- Clause **positive** literal -> **head - hd(c)**
- Clause **negated** literals -> **body - bd(c)**
- V_f -> countable set of FO variables (*A*, *B*, *C*, ...)
- V_h -> countable set of HO variables (*P*, *Q*, *R*, ...)

- **Generation** phase **-** $S_{PI}(\mathfrak{T}) = \{p_i \mid p_i \in ag_h(a) \wedge a \in ex(\mathfrak{T})\}$
  - **pruning** programs that contain external literals but lack clauses for their arguments

Example:

$$\begin{aligned}
&\mathtt{half_{1st}}(A,B)\text{:- }\mathtt{reverse}(A,C),\\
&\qquad\qquad\mathtt{case_{list}}(p_{[\,]}, p_{[H|T]}, C, B).\\
&\qquad\mathtt{p}_{[\,]}(A)\text{:- }\mathtt{empty}(A).\\
&\mathtt{p}_{[H|T]}(A,B,C)\text{:- }\mathtt{empty}(B), \mathtt{empty}(C).\\
&\mathtt{p}_{[H|T]}(A,B,C)\text{:- }\mathtt{front}(B,D)^4,\\
&\qquad\qquad\underline{\mathtt{case_{list}}(p_{[\,]}, p_{[H|T]}, D, E)},\\
&\qquad\qquad\mathtt{append}(E,A,C).
\end{aligned}$$

$$\begin{aligned}
&\mathtt{reverse}(A,B)\text{:- }\mathtt{empty}(C),\ \mathtt{fold}(p,C,A,B).\\
&\qquad\mathtt{p}(A,B,C)\text{:- }\mathtt{head}(C,B),\ \mathtt{tail}(C,A).
\end{aligned}$$

$$\begin{aligned}
&\mathtt{issubtree}(A,B)\text{:- }A = B.\\
&\mathtt{issubtree}(A,B)\text{:- }\mathtt{children}(A,C), \mathtt{any}(\mathtt{cond},C,B).\\
&\qquad\mathtt{cond}(A,B)\text{:- }\underline{\mathtt{issubtree}(A,B)}.
\end{aligned}$$

Definition 5 *Let L be a library, and $\mathfrak{T}$ an interpretable theory. $\mathfrak{T}$ is L-compatible if $\forall l \in ex(\mathfrak{T}), \exists! d \in L.$ s.t. $hd(d)\sigma = l$ for some substitution σ. Let $df(L, l) = d$ and $\theta(L, l) = \sigma$.*

- provides the clause from the library that is compatible with the external literal
- provides the specific substitution required to match the external literal with the clause in the library

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

Let $l = fold(p, C, A, B)$: $df(L, l) = \texttt{fold}(P, A, B, C)$ *and* $\theta(L, l) = \{P \mapsto p, A \mapsto C, B \mapsto A, C \mapsto B\}$.

- Clause **positive** literal -> **head - hd(c)**
- Clause **negated** literals -> **body - bd(c)**

Atoms s.t. $ag_h(l) \neq \emptyset$ *are external. The set of external atoms of an interpretable theory* $\mathfrak{T}$ *is denoted by* $ex(\mathfrak{T})$.

L-grounding

- When a definition takes more than one HO argument and arguments of instances partially overlap, duplicating clauses may be required during the construction of the L-grounding

- Soundness of the pruning mechanism is preserved

    - FO literals uniquely depend on the arguments fed to HO definitions

    - the pruning mechanism safely eliminates unnecessary clauses during the subsequent steps of theory evaluation and refinement

The system relies on the user to provide HO definitions (similar to MetagolHO)

- can also be in the form of templates, such as ho(P, Q, x, y):- P(Q, x, y)

$$\texttt{fold}(P, A, B, C)\text{:- }\texttt{empty}(B), C = A.$$
$$\texttt{fold}(P, A, B, C)\text{:- }\texttt{head}(B, H), \texttt{P}(A, H, D),$$
$$\texttt{tail}(B, T), \texttt{fold}(P, D, T, C).$$

Let $l = fold(p, C, A, B)$*:* $df(L, l) = \texttt{fold}(P, A, B, C)$ *and* $\theta(L, l) = \{P \mapsto p, A \mapsto C, B \mapsto A, C \mapsto B\}$.

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

Example 3 *Using the library of Example 2 and a modified version of the program from Section 2.1 (p is replaced by* $\texttt{fold}_{p\_a}$ *for clarity purposes), we get the following L-grounding:*

$$\texttt{reverse}(A, B)\text{:- }\texttt{empty}(C),\ \texttt{fold}_a(C, A, B).$$
$$\texttt{fold}_{p\_a}(A, B, C)\text{:- }\texttt{head}(C, B),\ \texttt{tail}(C, A).$$
$$\texttt{fold}_a(A, B, C)\text{:- }\texttt{fold}(\texttt{fold}_{p\_a}, A, B, C).$$
$$\texttt{fold}(P, A, B, C)\text{:- }\texttt{empty}(B), A = C.$$
$$\texttt{fold}(P, A, B, C)\text{:- }\texttt{head}(B, H), P(H, D),$$
$$\texttt{tail}(B, T), \texttt{fold}(P, D, T, C).$$

$\texttt{fold}_a(C, A, B)$ *replaces* $\texttt{fold}(\texttt{fold}_{p\_a}, C, A, B)$*. The first two clauses form the principal program.*

Interpretable Theories and Constraints

Definition 6 (Θ-subsumption) *An FO theory T_1 subsumes an FO theory T_2, denoted by $T_1 \leq_\theta T_2$ iff, $\forall c_2 \in T_2 \exists c_1 \in T_1$ s.t. $c_1 \leq_\theta c_2$, where $c_1 \leq_\theta c_2$ iff, $\exists \theta$ s.t. $c_1\theta \subseteq c_2$.*

Proposition 1 *if $T_1 \leq_\theta T_2$, then $T_1 \models T_2$*

The pruning ability of *Popper*'s Generalization and specialization constraints follows from Proposition 1.

Definition 7 *An FO theory T_1 is a* generalization *(*specialization*) of an FO theory T_2 iff $T_1 \leq_\theta T_2$ ($T_2 \leq_\theta T_1$).*

Given a library L and a space of L-compatible theories, we can compare L-groundings using Θ-subsumption and prune generalizations (specializations), based on the **Test phase**.

Groundings and Elimination Constraints

- **generate phase** -> elimination constraints prune **separable** programs
  - no head literal of a clause occurs as a body literal of a clause in the set
- L-groundings are **non-separable** and do not require pruning
  - Querying the ASP solver directly for L-groundings is **inefficient**
- query for the principal program, treating the definitions from the library as BK
  - reintroduces the rest of the L-grounding during the "test" phase
- To efficiently implement HO synthesis, we introduce **call graph constraints**
  - defines the relationship between HO literals and auxiliary clauses

Groundings and Elimination Constraints

Example 3 *Using the library of Example 2 and a modified version of the program from Section 2.1 (p is replaced by* $\texttt{fold}_{p\_a}$ *for clarity purposes), we get the following L-grounding:*

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

$\texttt{fold}_a(C, A, B)$ *replaces* $\texttt{fold}(\texttt{fold}_{p\_a}, C, A, B)$*. The first two clauses form the principal program.*

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

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

Negation, Generalization, and Specialization

$E^+ : f(b). \quad f(c).$ $\qquad$ $\mathbf{E}^- : f(a).$

$BK : \left\{ \begin{array}{ll} p(a). & p(b). \\ q(a). & q(c). \end{array} \right\}$ $\qquad$ $\mathbf{HO} : N(P, A){:}\text{-} \neg P(A).$

$\underline{prog_s}$

$$\boxed{\begin{array}{l} \mathtt{f}(A){:}\text{-}\ \mathtt{N}(p_1, A). \\ \mathtt{p_1}(A){:}\text{-}\ \mathtt{p}(A),\ \mathtt{q}(A). \end{array}}$$

$\underline{prog_f}$

$$\boxed{\begin{array}{l} \mathtt{f}(A){:}\text{-}\ \mathtt{N}(p_1, A). \\ \mathtt{p_1}(A){:}\text{-}\ \mathtt{p}(A). \end{array}}$$

$prog_f \models \neg f(b) \wedge \neg f(a) \wedge f(c)$

Negation, Generalization, and Specialization

$E^+ : f(a). \quad f(b).$

$\mathbf{E}^- : f(c). \quad f(d).$

$BK : \{ \ p(d). \quad q(c). \ \}$

$\mathbf{HO} : N(P, X) \text{:-} \neg P(X).$

$\underline{prog_s}$

$$
\begin{array}{|l|}
\hline
\ \ \mathtt{f}(A) \text{:-} \mathtt{N}(p_1, A). \\
\mathtt{p}_1(A) \text{:-} \mathtt{p}(A). \\
\mathtt{p}_1(A) \text{:-} \mathtt{q}(A). \\
\hline
\end{array}
$$

$\underline{prog_f}$

$$
\begin{array}{|l|}
\hline
\ \ \mathtt{f}(A) \text{:-} \mathtt{N}(p_1, A). \\
\mathtt{p}_1(A) \text{:-} \mathtt{p}(A). \\
\\
\hline
\end{array}
$$

$prog_f \models f(a) \land f(b) \land f(c)$

Task	*Popper* (**Opt**)	#Literals	PI?	*Hopper*	***Hopper* (Opt)**	#Literals	HO-Predicates	*Metagol*$_{HO}$	Metatypes?
Learning Programs by learning from Failures [Cropper and Morel, 2021a]									
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

Table 1: We ran *Popper*, *Hopper*, *optimized Hopper* , and *Metagol*$_{HO}$ on a single core with a timeout of 300 second. Times denote the average of 5 runs. Evaluation time for *Popper* and *Hopper* was set to a thousandth of a second, sufficient time for all task involved.

Conclusion

- Popper was extended to effectively incorporate HO definitions provided by the user during the learning process
- the optimized version of the extended system, **Hopper**, outperforms **Popper** on most tasks
- Hopper requires minimal guidance compared to **MetagolHO**
- Although it is confirmed that Hopper can theoretically find the solution, the successful invention of an HO predicate during learning has not been achieved yet
    - Future work

Thank you for your time!

Seminar in Knowledge Representation and Reasoning – 184.712 – 2023S

Simão Costa (e12202234)