Towards Efficient Higher-order Logic Learning in a First-order Datalog Framework

Niels Pahlavi and Stephen Muggleton

Department of Computing, Imperial College London,
180 Queen's Gate, London SW7 2BZ, UK
niels.pahlavi@imperial.ac.uk, s.muggleton@imperial.ac.uk

Abstract. Within ILP, the concepts to be learned are normally considered as being succinctly representable in first-order logic. In a previous paper the authors demonstrated that increased predictive accuracy can be achieved by employing higher-order logic (HOL) in the background knowledge. In this paper, the flexible higher-order Horn clauses (FHOHC) framework is introduced. It is more expressive than the formalism used previously and can be emulated (with the use of "holds" statements and flattening) in a fragment of Datalog. The decidability, compatibility with ILP systems like Progol and positive learnability results of Datalog are then used towards efficient higher-order logic learning (HOLL). We show with experiments that this approach outperforms the HOLL system λProgol and that it can learn concepts in other HOLL settings like learning HOL and using HOL for abduction.

1 Introduction

Within inductive logic programming (ILP), it is usual to assume that all concepts to be learned can be succinctly represented in first-order logic (FOL). However, in [11] the authors demonstrated that in certain learning problems increased predictive accuracy can be achieved by employing higher-order logic (HOL) in background knowledge, thus advocating higher-order logic learning (HOLL). In this paper we explore whether some of the learning advantages provided by a HOL framework can be achieved within FOL. In particular, we introduce and explore a HOL formalism called flexible higher-order Horn clauses (FHOHC). We also show that statements in FHOHC can be emulated in a fragment of Datalog FOL using "holds" statements (as suggested in [7]) and flattening (as defined in [12]). This fragment of FOL, called flattened holds Datalog programs (FHDP), has the advantage of being decidable and of having positive ILP learnability results. Figure 1 presents two examples of such HOL clauses (lines 1 and 3).

Using the power of expressivity of HOL in logic-based Machine Learning (thus realizing HOLL) to outperform first-order logic learning (FOLL) has been advocated in an ILP context, as in [11] and [4] but also with a different logic in [6]. Figure 2 summarizes three settings of interest for HOLL in ILP.

In [11], the HOLL system λProgol was introduced. It is based on the ILP system Progol [8], and its underlying logic is HOL as it is based on Miller and

Fig. 1. Flexible higher-order Horn clauses programs representing transitivity for binary relations and mathematical induction for Peano numbers (lines 1 and 3, respectively) and their corresponding flattened holds Datalog programs (lines 2 and 4, respectively)
$R(X,Y) \leftarrow transitive(R), R(X,Z), R(Z,Y)$. (1)
$holds(R,X,Y) \leftarrow holds(transitive,R), holds(R,X,Z), holds(R,Z,Y)$. (2)
$P(sko\_cst)$. $P(X) \leftarrow P(0), P(succ(sko\_cst))$. (3)
$holds(P,sko\_cst)$. $holds(P,X) \leftarrow holds(P,0), holds(succ,sko\_cst,Y), holds(P,Y)$. (4)

Fig. 2. HOLL settings in ILP

Setting	Background Knowledge	Hypothesis	Examples	Learning
1	HOL, Given	FOL, To be learned	FOL, Given	Induction
2	FOL, Given	HOL, To be learned	FOL, Given	Induction
3	HOL, Given	FOL/HOL, Given	FOL/HOL, To be learned	Abduction

Nadathur's Higher-order Horn Clauses (HOHC), defined in [9]. The paper experimentally compared λProgol with Progol in Setting 1 (see Figures 2 and 3). It was demonstrated that λProgol can achieve considerably higher accuracy in this setting than Progol, however several issues still need to be addressed. The HOL formalism HOHC was chosen for its supposed expressivity and soundness. Yet, several limitations are intrinsic to it. Clauses with flexible heads (atoms whose predicate is a variable [9]) are not allowed for decidability reasons, which limits the expressivity and may be a problem in Settings 2 and 3. There is an issue with complexity. The system has not yet been adapted to handle abduction as in Progol5 [8] and the Progol theoretical results remain to be proved for HOHC. We will see how the use of FHOHC and FHDP may overcome these issues.

In Section 2, the frameworks FHOHC and FHDP are described. Section 3 presents results in three different HOLL settings and develops the experiment detailed in [11]. Finally, Section 4 concludes and suggests further work.

2 HOLL with first-order Datalog and Progol

In Definition 1, we introduce the HOL formalism called flexible higher-order Horn clauses (FHOHC), which is based on first-order Horn clauses and allows for predicate (at least second-order) variables.

Definition 1. ***(Flexible Higher-order Horn Clauses (FHOHC))***.
A represents atomic formulas (or atoms), G goal formulas and D programme formulas (or definite formulas, or clauses). Horn clauses are defined by the following grammar. $G ::= A|G \wedge G$ *and* $D ::= A|G \supset A|\forall xD$. *An atomic formula is* $P(t_1,...,t_k)$ *where P is a either a predicate symbol or higher-order variable of arity k and* $t_1,...,t_k$ *are terms. A term is either a variable or* $f(t_1,...,t_j)$ *where f is a functor of arity j and* $t_1,...,t_j$ *are terms. A functor of arity* 0 *is a constant.*

Compared with HOHC [9], FHOHC allows for flexible heads and therefore for more expressivity. These were prevented in HOHC because of decidability issues

but we will now show how FHOHC can be emulated with a fragment of first-order Datalog. Datalog [1] is a restriction of Logic Programming that allows only variables and constants as terms (and hence avoids the use of function symbols). It has the advantage of being decidable. It has a declarative semantics and one can benefit from some positive ILP learnability results within it, as summarized in [5] and [2]. The use of "holds" statements as suggested in [7] allows us to turn a higher-order atom into a first-order one. Moreover the flattening/unflattening procedures [12] can translate generic Horn clauses into Datalog ones and vice-versa, without loss of generality. This is why we introduce the flattened holds Datalog programs (FHDP) in Definition 2 to emulate HOL and FHOHC.

Definition 2. ***(Flattened Holds Datalog Programs (FHDP))***.
A flattened holds Datalog program is a flexible higher-order Horn clause program which has been transformed as follows. First, every atom $P(t_1, ..., t_k)$*,* P *being a predicate symbol or a higher-order variable, is replaced by the atom* $holds(P, t_1, ..., t_k)$ *of arity* $k+1$*. Then the flattening algorithm, defined in [12], is applied to the modified program.*

In Figure 1, two examples of such FHDP programs are presented (lines 2 and 4). With such an underlying framework, we can obviously use any of the first-order ILP systems available. Progol [8] is a popular implementation, which allows us to learn in the HOLL Settings 1 and 2 requiring inductive reasoning but also in Setting 3 requiring abductive reasoning with Progol5 (see Figure 2). Developing HOLL with the Datalog fragment FHDP and Progol enables us to directly use an ILP system like Progol and its results (including Progol5), to benefit from the efficiency of deduction of Datalog and its decidability, and to have more expressivity than HOHC with flexible heads to handle more learning settings. In terms of learnability and predictability, the existing positive results for Datalog can be used but the higher-order nature of the programs may also provide more complex and better choices of features, as analysed in [2].

3 Experiments

In this section, we show how our Datalog approach can be applied on three examples covering the three HOLL settings defined in Section 1. All the corresponding files and experiments can be found at [10]. In Examples 1,2 and 3, (...) corresponds to omitted parts.

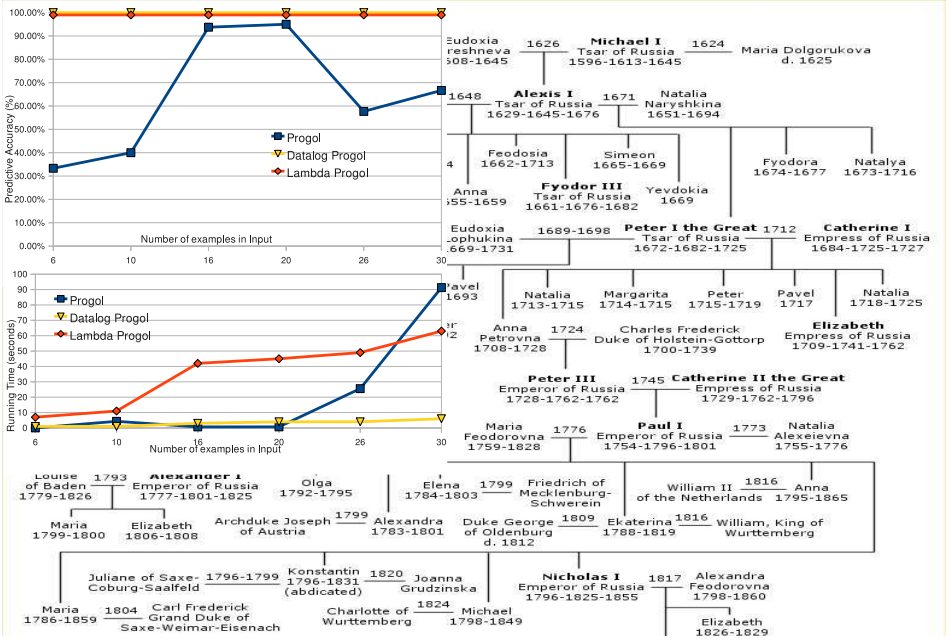
HOLL Setting1: Inductive learning of FOL hypothesis with HOL background .
This follows the experiment fully described in [11], about the learning of the predicate *ancestor* given the predicate *parent*. Progol rarely finds the definition (either returning incorrect recursive definitions, non-recursive definitions or not being able to induce clauses that compress the data). On the other hand, λProgol learns the correct definition in all the cases, which is non recursive and can be learned from any given positive example. This is due to the presence of the higher-order predicate *trans_closure*, which represents the transitive closure of

any binary relation. Here we use our FHDP approach in the comparison. The λProgol files (see Example 1) are totally emulated (with the exception of the addition of a prune statement to prevent higher-order tautologies in Progol). Hence the same learned hypothesis and the same predicative accuracy results (see Figure 3). In Figure 3, the running times are also added, which show that the FHDP approach is considerably faster compared to standard Progol and λProgol (both being similar), which illustrates the efficiency of the Datalog framework. This type of learning can be used with multiple higher-order predicates and with non-IID problems.

Example 1. **Setting1: Input file for learning ancestor.**

:- modeh(*,holds(ancestor,+person,+person))?
:- modeb(*,holds(#predso,#predpp,+person,+person))?
predso(trans_clos). predpp(parent). predpp(married).
person(X) :- male(X). person(X) :- female(X).
holds(trans_clos,R,X,Y) :- holds(R,X,Y).
holds(trans_clos,R,X,Z) :- holds(R,X,Y), holds(trans_clos,R,Y,Z).
prune(holds(P,A,B),Body) :- in(holds(trans_clos,P,A,B),Body).
holds(married,michael_I,eudoxia_stresh). (...) holds(parent,michael_I,alexis_I). (...)
:-holds(ancestor,maria_1,nat_narysh). (...) holds(ancestor,alex_II,maria_6). (...)
Learned clause: holds(ancestor,X,Y) :- holds(trans_clos,parent,X,Y).

Fig. 3. Left: Comparison between Progol, Progol with FHDP and λProgol on the Ancestor example (upper graph: predictive accuracy; lower graph: running times). Right: Part (around one third) of the Romanov dynasty tree used in the experiment



HOLL Setting2: Inductive learning of HOL hypothesis with FOL background knowledge. In Example 2, a higher-order clause representing the transitivity of any binary relation (as in Figure 1) is learned from examples of two binary relations (one being transitive, the other not). The running time is under a second. This learning could not be done with λProgol, as it involves a clause with a flexible head. This type of learning can be used for transfer learning.

Example 2. **Setting2: Input file for learning transitivity.**

```
:- modeh(*,holds(+predicate,+argument,+argument))?
:- modeb(*,holds(#predicate,+predicate))?
:- modeb(*,holds(+predicate,+argument,+argument))?
predicate(trans). predicate(cause). predicate(pred). argument(a). (...)
holds(trans,cause). :- holds(trans,pred).
holds(cause,a,b). (...) holds(pred,c,d). (...)
:- holds(cause,b,a). (...) :- holds(pred,a,c). (...)
Learned clause: holds(R,X,Y) :- holds(trans,R),holds(R,X,Z),holds(R,Z,Y).
```

HOLL Setting3: Abductive learning of FOL hypothesis with HOL background knowledge. In Example 3, we follow the approach in [3], to formulate and adapt the general (second-order) concept of mathematical induction for Peano numbers $(f(0) \wedge (f(x) \rightarrow f(Sx)) \rightarrow f(y))$ in the FHOHC and FHDP frameworks (as in Figure 1). It is included with the *less_than* predicate and is used to abduce the "base case" of a particular (first-order) predicate f. We also have to include the Clark completion of the "step case" of the definition of f for mathematical induction to be utilized. The running time is under five seconds. This learning can be adapted to structural induction, to predicate invention and to abduce higher-order hypothesis.

Example 3. **Setting3: Input file for abduction with mathematical induction.**

```
:-modeh(*,holds(lt,#peano_int,+peano_int))?
:-modeb(*,holds(s,+peano_int,+peano_int))? :-observable(holds/2)?
peano_int(0). peano_int(s(X)) :- peano_int(X). holds(s,W,s(W)).
holds(F,sko_x). holds(F,X) :- holds(F,0),holds(s,sko_x,Y),holds(F,Y).
holds(lt,U,V) :- holds(s,X,U),holds(s,Y,V),holds(lt,X,Y).
holds(f,X) :- holds(s,X,Y),holds(lt,X,Y). holds(lt,X,Y) :- holds(s,X,Y),holds(f,X).
holds(f,s(s(s(0)))). (...) :- holds(lt,s(0),0). (...)
Learned clause: holds(lt,0,Y) :- holds(s,X,Y).
```

4 Conclusion and Further Work

In this paper, the HOL framework FHOHC is introduced, which is more expressive than HOHC and can be emulated (with the use of "holds" statements and flattening) in the FHDP fragment of Datalog, which is decidable, efficient, can be directly used by ILP systems like Progol and has positive learnability results. We have showed on concrete experiments that this approach learns as well as λProgol (based on HOHC) on HOLL Setting1 (learning of FOL with HOL background) but with better running times. Moreover, it can learn examples in HOLL Settings 2 (learning of HOL with FOL background) and 3 (abduction

of FOL with HOL background), in which λProgol can not be used or has not yet been implemented to learn. We are currently finishing more experiments in order to have more insight on the performances of this new approach (including one about the learning of the move of a bishop in chess involving multiple HOLL settings). We are also completing the formalization of the equivalence between FHOHC and FHDP. These could be both included in a longer version of the paper. We think that this approach could be used further, including in more complex situations, to abduce HOL, for predicate invention and for transfer learning.

References

1. S. Ceri, G. Gottlob, and L. Tanca. What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1:146–166, 1989.
2. W. Cohen and C.D. Page. Polynomial learnability and Inductive Logic Programming: methods and results. *New Generation Computing*, 13:369–409, 1995.
3. J.L. Darlington. Automatic Theorem Proving with Equality Substitutions and Mathematical Induction. In *Machine Intelligence 3*, pages 113–127. 1968.
4. C. Feng and S.H. Muggleton. Towards inductive generalisation in higher order logic. In *Proc. Ninth Int. Work. on Machine Learning*, pages 154–162, 1992.
5. Jörg-Uwe Kietz and Saso Dzeroski. Inductive Logic Programming and Learnability. *SIGART Bulletin*, 5(1):22–32, 1994.
6. J.W. Lloyd. *Logic for Learning*. Springer, Berlin, 2003.
7. John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In *Machine Intelligence 4*. 1969.
8. S. Muggleton and C. Bryant. Theory completion using inverse entailment. *10th Intern. Workshop on Inductive Logic Programming (ILP-00)*, pages 130–146, 2000.
9. G. Nadathur and D. Miller. Higher-order Horn Clauses. *Journal of the ACM*, 1990.
10. N. Pahlavi. ILP11 Experiments. http://www.doc.ic.ac.uk/ namdp05/ilp11.
11. N. Pahlavi and S.H. Muggleton. Can HOLL Outperform FOLL? In *Proceedings of the 20th Intern. Conf. on Inductive Logic Programming*. Springer-Verlag, 2010.
12. Céline Rouveirol. Flattening and Saturation: Two Representation Changes for Generalization. *Machine Learning*, 14(1):219–232, 1994.