Learning First Order Universal Horn Expressions

Roni Khardon*
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ
Scotland
roni@dcs.ed.ac.uk

Abstract

The problem of learning universally quantified function free first order Horn expressions from equivalence and membership queries is studied. The results presented generalise the known algorithm for propositional Horn expressions [AFP92] for several models of learning in first order logic. It is shown that exact learning is possible with membership and equivalence queries with resources polynomial in the number of clauses in the expression, though superpolynomial in the number of universally quantified variables. Similar results for related models including entailment queries and ILP are also derived.

1 Introduction

We study the problem of exactly identifying first order Horn formulas using Angluin's [Ang88] model of exact learning. Much of the work in learning theory has dealt with learning of Boolean expressions in propositional logic. Early treatments of relational expressions appear in [Val85, Hau89], but only recently more attention was given to the subject in the form of Inductive Logic Programming (ILP) (see e.g. [MDR94, Coh95a, Coh95b]), and many results on learning from examples were derived. It is clear that the relational learning problem is harder than the propositional one and indeed except for very restricted cases it is computationally hard [Coh95b]. To tackle this issue in the propositional domain various queries and oracles that allow for efficient learning have been studied [Val84, Ang88]. In particular, propositional Horn expressions are known to be learnable in polynomial time from equivalence and membership queries [AFP92], and from entailment queries [FP93]. In the relational domain, some forms of queries have been used in several systems (e.g. [Sha83, DRB92]) and results on learnability in the limit were derived. More recently Reddy and Tadepalli [RT97] considered the use of membership queries and have shown that Horn definitions (where all clauses have the same unique positive literal) are learnable.

In this paper we show that function free universally quantified Horn expressions are exactly learnable. Our method follows closely the ideas in [AFP92] and generalises the result by finding appropriate first order constructs. To illustrate the results and parameters (exact definitions for the various notions appear in the next section) consider the Horn expression

$$\forall x_1, x_2, x_3, \\ (p_1(x_1, x_2)p_2(x_1, x_3) \rightarrow p_1(x_2, x_1)) \wedge \\ (p_3(x_3, x_1)p_1(x_3, x_1) \rightarrow p_4(x_3)).$$

The language includes $|P| = 4$ predicates $p_1, \ldots, p_4$ each of arity at most $a = 2$, and the expression has $k = 3$ universally quantified variables, and $m = 2$ clauses. Our algorithm learns this class of expressions with query complexity polynomial in $m, |P|, k^a, k^k, n$ where n is the number of objects in the examples it sees, and time complexity polynomial in the above parameters and n^k.

In deriving the results we use a variant of the standard semantics where each universally quantified variable in an expression must be bound to a unique element. This setting has been considered before by Haussler [Hau89]. Our main result is that in this setting the number of equivalence queries is polynomial in $m, |P|, k^a, n$ whereas the running time and membership queries are as above. In fact our result can also be seen as extending Haussler's positive result (that shows the learnability of a single clause) in having more than one clause in the expression though restricting the clauses to be Horn. While this model has its own merits we derive other results for the standard model, entailment queries, and ILP as corollaries. With entailment queries the running time does not grow with n^k. Some of these extensions hold for a more expressive language allowing an arbitrary number of equalities in the clauses, as in

$$(p_2(x_1, x_3)p_3(x_2, x_1) \rightarrow p_4(x_2) \vee (x_1 = x_2))$$

thus going somewhat beyond the pure Horn case.

Our learning algorithm uses *pairings* of examples, a variant of direct products that have been used before in similar contexts (see e.g. [HST97]). Using pairings the progress of

*Part of this work was done while the author was at Harvard University and supported by ARO grant DAAL03-92-G-0115 and ONR grant N00014-95-1-0550.

the algorithm can be controlled so as to achieve the above bounds.

We also show that in some sense these results are not surprising. One can obtain similar bounds for a subset of the language just by using an appropriate simulation of the propositional algorithm in the first order domain. The direct result is however stronger and naturally more appealing.

The rest of the paper is organised as follows. Section 2 defines the learning problem and Section 3 presents some simple observations used throughout the paper. Section 4 shows how the propositional algorithm can be used in the first order domain. Section 5 discusses the use of direct products, Section 6 presents the main result on learning with the special semantics, and Section 7 extends this result for other models. In Section 8 we conclude with a brief discussion.

2 Preliminaries

2.1 First Order Horn Expressions

We consider a subset of the class of universally quantified expressions in first order logic. The learning problems under consideration will assume a pre-fixed known and finite signature of the language. That is, a finite set of predicates P each with its associated arity is fixed (we do not allow constants or other function symbols and thus these are not needed). In addition a set of variables $x_1, x_2, x_3, \ldots$ is used to construct expressions. First order universally quantified expressions over P are to be learned.

For definitions of first order languages see e.g. [CK90, Llo87]; here we briefly introduce the necessary constructs. A positive literal is predicate applied to a set of variables, that is, $p(X)$ where $p \in P$ and X is a set of variables of an appropriate size (the arity of p). A negative literal is obtained by adding the negation symbol to a positive literal, e.g. $\overline{p(X)}$. A clause is a disjunction of literals where all variables in the clause are (implicitly) universally quantified. A Horn clause has at most one positive literal, and a Horn expression is a conjunction of Horn clauses. Note that any clause can be written as $c = (\wedge_{n \in \text{Neg}} n) \rightarrow (\vee_{p \in \text{Pos}} p)$ where Neg and Pos are the sets of negative and positive literals of C respectively. When doing so we will refer to $(\wedge_{n \in \text{Neg}} n)$ as the antecedent of c and to $(\vee_{p \in \text{Pos}} p)$ as the consequent of c. We will make use of several assumptions and restrictions on expressions.

A1: The expressions are function free (and constant free).
A2: The arity of all predicates in P is bounded by a (small) constant a.
A3: Every clause has at most one positive literal (i.e. a Horn clause).
A4: All clauses under consideration are "non-generative" in the following sense. Let $c = (\wedge_{n \in \text{Neg}} n) \rightarrow (\vee_{p \in \text{Pos}} p)$ be as above, then every variable that appears in Pos also appears in Neg, so that the rule does not generate knowledge on objects not tested in its condition, hence the name above.
A5: Every clause must have at least one negative literal.

Let $\mathcal{F}(P)$ be the set of all expressions satisfying assumptions **A1, A2, A3**, and let $\mathcal{F}(P)^-$ be the set of all expressions satisfying the assumptions **A1, A2, A3, A4**. (Note that when **A1** is assumed, **A4** implies **A5**). For example $p(x, y) \rightarrow q(x)$, is in $\mathcal{F}(P)^-$, $p(x, y) \rightarrow q(z)$ and $q(x)$ are in $\mathcal{F}(P)$ but not in $\mathcal{F}(P)^-$ (violate A4), $p(x, y) \rightarrow q(x) \vee q(y)$ is in neither (violates A3).

Finally, let $\mathcal{F}(P, =)$ be the language $\mathcal{F}(P)$ extended so that clauses can have any number of literals of the form $(x_i = x_j)$, or $(x_i \neq x_j)$ where x_i, x_j are variables in the clause. An example clause appears in the introduction. Hence, $\mathcal{F}(P, =)$ goes somewhat beyond Horn expressions (if equalities are considered as positive literals).

2.2 Examples

An example is an interpretation I of the predicates in P [Llo87]. It lists a set of domain elements and the truth values of all instantiations of predicates on these elements. The *extension* of a predicate in I is the set of positive instantiations of it that are true in I. The extension of an interpretation is the set of positive instantiations of predicates in P that are true in it. Assumption **A2** implies that the size of the extension of an example is polynomial in the number of domain elements.

Examples of this form have been used in [Hau89] and are motivated by the scenario of acting in structural domains (e.g. [Kha96, RTR96, RT97]). They are also used in the non-monotonic form of ILP [DRD94]. In structural domains, domain elements are objects in the world and an instantiation describes properties and relations of objects. We therefore refer to domain elements as objects. For convenience we assume a standard way of naming objects (and think of them as a list of natural numbers).

For example, for the language of expressions given in the introduction, I may have the extension $\{p_1(1,2), p_1(3,5), p_2(1,5), p_4(2)\}$ for the set of objects $\{1,2,3,4,5\}$. Notice that no positive fact holds in I for the object 4.

2.3 Semantics

Note that the classes of expressions were defined syntactically. We attach a concept to each expression by defining appropriate semantics. Since the paper discusses two different semantics, an expression may be mapped to two different concepts under these. When the chosen semantics is not clear from the context we would specify which concept is meant. For the meantime we define a single semantics, the standard one [CK90, Llo87].

Let $l(X)$ be a literal, I an interpretation and θ a mapping of the variables in X to objects in I. The *ground literal* $l(\theta(X))$ is obtained from $l(X)$ by substituting variables in it according to θ. A ground positive literal $p(\theta(X))$ is true in I if and only if it is in the extension of the relevant predicate. A ground equality literal $\theta(x_i = x_j)$ is true in I if and only if θ maps x_i and x_j to the same objects. A ground negative literal is true in I if and only if its negation is not.

A clause $C \in \mathcal{F}(P)$ is true in an interpretation I if for all substitutions θ of variables in C to objects in I at least one of the literals in $C(\theta)$ is true in I. An expression $f \in \mathcal{F}(P)$ is true in I if all clauses C in f are true in I. The terms (1) f is true in I, (2) I is a positive example for f, (3) I satisfies f, (4) I is a model of f, and (5) $I \models f$, have the same meaning. Let $f, g \in \mathcal{F}(P)$ then f implies g, denoted $f \models g$, if every model of f is also a model of g.

2.4 Parametrising the Concept Class

The languages defined above can be further parametrised by restricting the number of (universally quantified) variables in each clause. Denote the respective classes where the number of variables is bounded by k, by $\mathcal{F}^k(P), \mathcal{F}^k(P)^-$, and $\mathcal{F}^k(P,=)$.

For $f \in \mathcal{F}^k(P,=)$, one can test whether $I \models f$ by enumeration in time $O(n^k)$ if I has n objects. In general even evaluating a single clause on a single interpretation is NP-Hard (see e.g. [Val94] pp. 212) if k is not bounded. Recent results [PY97] suggest that it is not likely to have an algorithm polynomial in n even for small non-constant values of k. We will thus assume that k is constant whenever such evaluation needs to be performed. Note that this restriction does not limit the size of clauses to be constant but instead longer clauses must reuse the same variables repeatedly. Similar restrictions have been previously used by Haussler [Hau89].

Other assumptions that ensure tractability have also been used (e.g. determinacy in [DMR92]); we do not address such restrictions here.

2.5 The Learning Model

The learning model uses several forms of queries [Ang88, FP93]. Let $\mathcal{F}$ be a class under consideration, and let $f \in \mathcal{F}$ be the target function. For *Equivalence Queries* the learner presents a hypothesis $H \in \mathcal{F}$ and the oracle return "yes" if $H = f$ and otherwise it returns an interpretation I that is a counter example ($I \models f$ and $I \not\models H$ or vise versa). For *Membership Queries* the learner presents an interpretation I and the oracle return "yes" iff $I \models f$. For *Entailment Queries*, the learner presents $C(\theta(X))$, a ground instance of a clause $C \in \mathcal{F}$ (i.e. all variable are substituted to objects) and the oracle return "yes" iff $f \models C$. Further definitions for the ILP setting are given in Section 7.

In the learning model a target function $f \in \mathcal{F}$ is fixed and hidden from the learner. A learner interacts with the oracles and has to find an expression H that is equivalent to f with respect to $\models$.

3 Small Interpretations

The following lemmas indicate that we can always restrict our attention to small interpretations. Let I be an interpretation, and let A be a subset of the objects in I. Then $I_{|A}$ is the interpretation induced from I by deleting the objects not in A and all the instantiated predicates on these objects. Let $\mathcal{I}^k$ be the set of interpretations where the number of objects is at most k.

Lemma 3.1 *Let $f \in \mathcal{F}^k(P)$ and let I be any interpretation. If $I \not\models f$ then there is a set A of objects of I such that*
(1) $|A| = k$, $I_{|A} \in \mathcal{I}^k$, and $I_{|A} \not\models f$
(2) $\forall B \supset A$, $I_{|B} \not\models f$.

Proof: This follows since to falsify f a single substitution θ is sufficient and since f has at most k variables it is sufficient to include in A the objects mentioned in θ. Clearly, any superset B of A can falsify f using the same θ. ■

Lemma 3.2 *Let $f \in \mathcal{F}(P)$, and let I be any interpretation. If $I \models f$ then for any set A of objects of I, $I_{|A} \models f$.*

Proof: Assume $I_{|A} \not\models f$. Then there is a substitution θ and a clause C in f such that C is not true in $I_{|A}$. Clearly C is not true in I under the same θ. ■

We next show that if the domain is fixed then $\mathcal{F}(P)^-$ can be simulated by propositional expressions. In order to relate interpretations to the standard propositional setting we assume a fixed number of objects k, and object names $1, 2, \ldots, k$. For each predicate $r()$ of arity a we create k^a propositional variables $r_{(1,\ldots,1)}, \ldots, r_{(k,\ldots,k)}$, corresponding to all instantiations of $r()$ over objects in $1, 2, \ldots, k$. An interpretation I corresponds to an assignment of values in $\{0, 1\}$ to the propositional variables in a natural way. Namely, for a tuple A of a objects in $1, \ldots, k$, the propositional variable r_A is assigned 1 if and only if $r(A) \in I$. When discussing propositional expressions and the propositional learning algorithm we implicitly assume that this translation is used.

Let f be a universally quantified Horn expression on a set of variables $X = (X_1, \ldots, X_k)$

$$f = \forall X,\ C_1(X)C_2(X)\ldots C_m(X).$$

Let $\theta_1, \ldots, \theta_{k^k}$ be an enumeration of all possible mappings of k object variables to objects in an interpretation with exactly k objects. Consider the propositional expression

$$\begin{aligned} f_p \quad = \quad & C_1(\theta_1(X))C_1(\theta_2(X))\ldots C_1(\theta_{k^k}(X)) \\ & C_2(\theta_1(X))C_2(\theta_2(X))\ldots C_2(\theta_{k^k}(X)) \\ & \ldots \\ & C_m(\theta_1(X))C_m(\theta_2(X))\ldots C_m(\theta_{k^k}(X)). \end{aligned}$$

For $I \in \mathcal{I}^k$ define $inflate(I)$ to be the interpretation with the same extension as I but where the number of objects is exactly k. Namely to get $inflate(I)$ we add new "phantom" objects to I. We have the following:

Lemma 3.3 *Let $f \in \mathcal{F}^k(P)^-$, $I \in \mathcal{I}^k$, and let f_p be the propositional version of f described above. Then the following conditions are equivalent:*
(1) $I \not\models f$
(2) $inflate(I) \not\models f$
(3) $inflate(I) \not\models f_p$.

Proof: Clearly (1) implies (2) and (3) since the falsifying substitution in I suffices. Now (3) implies (2) since the clause falsified in f_p supplies the falsifying substitution in f. To see that (2) implies (1) notice that phantom assignments do not change the truth value. For any θ that maps a variable to a phantom object, and any clause C that uses this variable, $C(\theta(X))$ is true since the antecedent of C is false. This is guaranteed by assumption **A4** since if a variable appears in a clause it must appear in the antecedent at least once. ■

4 Using the Propositional Algorithm

In this section we present a simulation result showing that the propositional algorithm from [AFP92] can be adapted to learn $\mathcal{F}(P)^-$.

4.1 The Algorithm Prop-Horn

We describe the propositional algorithm which we refer to later as Prop-Horn. We first define the basic operations of

1. Maintain an ordered set of interpretations S, initialised to $\emptyset$ and let $H = candidates(S)$.
2. Repeat until $H = f$:
   - Ask an equivalence query using H, to get a counter example in case $H \neq f$.
   - On a positive counter example remove wrong clauses from H.
   - On a negative counter example I:
     For $i = 1$ to m (where $S = \{s_1, \ldots, s_m\}$)
       If $J = s_i \wedge I$ is negative (use MQ), and its extension is smaller than that of s_i
       then replace s_i with J, and quit loop.
     If no s_i was replaced then add I as the last element of S.
     After each negative counter example, recompute H as $candidates(S)$.

Figure 1: The algorithm Prop-Horn

the algorithm. Let I be an interpretation, and let $ant(I)$ be the conjunction of all positive ground literals true in I, and $neg(I)$ be the set of all positive ground literals that are false in I. The set $candidates(I)$ is the set of clauses $\{ant(I) \rightarrow A \mid A \in neg(I)\} \cup \{\overline{ant(I)}\}$. For a set of interpretations S, $candidates(S) = \cup_{s \in S} candidates(s)$.

Let I_1, I_2 be interpretations with the same set of objects. The *intersection* of I_1, I_2 is defined to have the same objects as in I_1, I_2, and its extension is defined to have the intersection of the positive ground literals in I_1, I_2. We denote the intersection by $I_1 \wedge I_2$. Note that while the class of universal sentences is oblivious to naming of objects the propositional treatment here is sensitive in this respect.

The algorithm is described in Figure 1. The algorithm maintains an ordered set of "representative" negative examples from which it builds its hypothesis by using the $candidates()$ operation. A new negative counter example either removes a wrong clause, refines one of the current representative examples, or is otherwise added as a new representative example. The correctness and efficiency of the algorithm follow by showing that no two representative examples falsify the same clause in the representation for f, and that each refinement makes progress is some measurable way [AFP92].

4.2 The Algorithm FOL-Horn-1

The observations in the previous section suggest a method for learning $\mathcal{F}(P)^-$ by using the propositional algorithm. The learning algorithm will use the propositional hypothesis of Prop-Horn and will adapt the number of objects in counter examples to be exactly k by using membership queries to reduce the number of objects (relying on Lemma 3.1 and Lemma 3.2) or using $inflate()$ to set the number of objects to k (relying on Lemma 3.3). This however does not quite work if arbitrary examples rather than examples in $\mathcal{I}^k$ are used since f_p is not guaranteed to be correct on these.

We next show that this difficulty can be overcome by adapting the algorithm to use a first order hypothesis. We call the modified algorithm FOL-Horn-1.

The algorithm FOL-Horn-1 runs Prop-Horn using $\mathcal{I}^k$ as the domain and simulating its oracles while interacting with the first order oracles. Assume first that $f \in \mathcal{F}^k(P)^-$ and the algorithm knows the correct value of k.

The algorithm uses Prop-Horn's set of interpretations S to generate its own hypothesis as follows. For an interpretation I, $ant(I)$ is a conjunction of positive literals obtained by listing all the positive literals in I and replacing each object with a distinct variable. Let X be the set of variables in $ant(I)$. Then $candidates(I)$ includes the set of Horn clauses of the form $ant(I) \rightarrow p(Y)$, where p is a predicate in the language and Y is a subset of X of the appropriate arity. In addition, the set $candidates(I)$ also includes the clause $\overline{ant(I)}$ (i.e. the one with the empty consequent). The hypothesis of the algorithm is generated as $H = \wedge_{s_i \in S} candidates(s_i)$. Initially $S = \emptyset$ and H is true on any interpretation.

When Prop-Horn asks a membership query (by intersecting x with the elements of S) the queries are passed directly to the membership oracle and answered in the same way.

When Prop-Horn asks an equivalence query the algorithm recomputes H as $H = \wedge_{s_i \in S} candidates(s_i)$ and asks an equivalence query. Given a positive counter example the algorithm evaluates all clauses in H on it, and removes any clause falsified by I from H. Evaluation of a single clause on I can be done by iterating over all possible substitutions, hence in time $O(n^k)$ where n is the number of objects in I.

Given a negative counter example if it has more than k objects the algorithm first finds a subset of objects that is sufficient as a counter example. This can be done greedily by removing one object at a time and asking a membership query. By Lemma 3.1 and Lemma 3.2 this yields a correct counter example that has at most k objects. Let I be the minimal counter example found; the algorithm renames the objects of I using names in $\{1, 2, \ldots, k\}$, and presents $x = inflate(I)$ to Prop-Horn as a counter example.

Note that Prop-Horn's hypothesis is never evaluated (and hence need not be generated). Its computation is restricted to computing intersections and asking membership queries. These in fact can be incorporated into FOL-Horn-1. The algorithm can be adapted for the case when the value of k is not known. This is discussed in the proof of the following theorem that appears in the appendix.

Theorem 4.1 *The class $\mathcal{F}(P)^-$ is learnable by the algorithm FOL-Horn-1 using equivalence queries and membership queries. For $f \in \mathcal{F}^k(P)^-$ with m clauses, the number of queries is polynomial in $m, |P|, k^a, k^k, n$, and the time complexity is polynomial in the above parameters and n^k, where n is the largest number of objects in the counter examples.*

This result is improved upon in Section 7 where $\mathcal{F}(P,=)$ is shown to be learnable (with slightly better bounds). The result however may be useful for other possible generalisations.

5 Direct Products

Direct Products are a generalisation of the intersection operation appropriate for first order Horn expressions. Products are in some sense the correct notion since they characterise Horn expressions just as intersections characterise propositional Horn expressions (and intersections fall as a special case of products for propositional logic). Products are closely related to least general generalisations [Plo70] as pointed out e.g. in [HT95, HST97]. Unfortunately, products are difficult to handle since they result in an exponential growth in hypothesis size, and in addition make it hard to identify when "progress" is made as done in the propositional algorithm. We briefly introduce these ideas and problems that in turn serve to motivate our next result.

Let $I_1, I_2, \ldots, I_j$ be interpretations. The direct product of $I_1, I_2, \ldots, I_j$ denoted $\otimes(I_1, I_2, \ldots, I_j)$ is an interpretation. The set of objects in $\otimes(I_1, I_2, \ldots, I_j)$ is the set of tuples $(a_1, a_2, \ldots, a_j)$ where a_i is an object in I_i. The extension of $\otimes(I_1, I_2, \ldots, I_j)$ is defined as follows. Let P be a predicate of arity l and let $(c_1, \ldots, c_l)$ be a l-tuple of elements of $\otimes(I_1, I_2, \ldots, I_j)$, where $c_i = (a_{i_1}, a_{i_2}, \ldots, a_{i_j})$. Then $P(c_1, \ldots, c_l)$ is true in $\otimes(I_1, I_2, \ldots, I_j)$ if and only if for all $1 \leq q \leq j$ $P(a_{1_q}, a_{2_q}, \ldots, a_{l_q})$ is true in I_q. In words, $P(c_1, \ldots, c_l)$ is true if and only if component-wise P is true on the original tuples generating $(c_1, \ldots, c_l)$ in the corresponding interpretations. When $j = 2$ we also denote $\otimes(I_2, I_2)$ by $I_1 \otimes I_2$.

For example let $I_1 = \{p(1), p(2), q(1,2)\}$, and $I_2 = \{p(3), q(3,3), q(4,3)\}$, then the domain of $\otimes(I_1, I_2)$ is $\{(1,3), (1,4), (2,3), (2,4)\}$ and

$$\begin{aligned} \otimes(I_1, I_2) \quad = \quad & \{p((1,3)), p((2,3)), q((1,3),(2,3)), \\ & q((1,4),(2,3))\}. \end{aligned}$$

Renaming the objects by $\{5,6,7,8\}$ (in the same order) we get $\otimes(I_1, I_2) = \{p(5), p(7), q(5,7), q(6,7)\}$.

Products are important since they exactly characterise the class of Horn expressions. The following theorem is due to McKinsey [McK43] (see also [Hor51, CK90]). Our proofs for Lemma 6.1 and Lemma 6.3 below, follow similar lines.

Theorem 5.1 [McK43] *A universally quantified first order expression is equivalent to a universally quantified first order Horn expression if and only if its set of models is closed under direct products.*

The fact that models of Horn sentences are closed under products is used by the algorithm Prop-Horn when deciding whether to merge a counter example into the current s_i or not. If the intersection of s_i and a counter example I is negative then they both "cover" the same clause of f. Also, if both are falsified by the same clause then their intersection is guaranteed to be negative. In fact as will be seen below slightly more subtle properties are used though they rely on this property.

It is therefore natural to try and generalise the algorithm Prop-Horn by simply replacing the intersection operation by the direct product, thus lifting it to the relational level. Several difficulties however arise when this is done. Firstly, the size of $\otimes(I_1, \ldots, I_j)$ is exponential in j so we may end up with large interpretations. Following Lemma 3.1 and Lemma 3.2 one can try to reduce the size of interpretations dealt with, simply by omitting objects from them. This leads to further difficulties, mainly due to the fact that when evaluating a clause on an interpretation, several variables in the clause may bind to the same object. Such an object can be expanded in a product and thus the size of the extension found cannot be used to distinguish whether progress is made or not. These aspects are illustrated in the following examples where a naive generalisation of the algorithm is discussed. Recent results [RT98] show that it may be possible to overcome these problems if the Horn expression is acyclic.

Example 1 Let

$$f = \forall x, y,\ (P(x,y) \rightarrow Q(x)) \wedge (R(x,y) \rightarrow S(x))$$

and let $s_1 = \{P(a,b), P(b,c)\}$, and $I = \{P(1,1), R(1,2), Q(1)\}$. Then $s_1 \otimes I = \{P(a1, b1), P(b1, c1)\}$ is negative for f. It does include more objects, namely, $a2, b2, c2$ (for which no positive facts appear in the product). When these are removed we get an interpretation that is isomorphic to s_1.

Example 2 This example shows that it may be the case that I falsifies C, s_i "covers" C but all negative subsets of $J = s_i \otimes I$ do not have a smaller extension than s_i. This should be contrasted with Lemma 4 in [AFP92] and Lemma 6.4 below. It also shows that this case may be confused with the situation in Example 1. Consider

$$\begin{aligned} f \quad = \quad & \forall x, y, z, \\ & (P(x,y)Q(y,z) \rightarrow S(x)) \wedge \\ & (Q(x,y)P(y,z) \rightarrow S(x)). \end{aligned}$$

Let

$$I_1 = \{P(1,2), P(2,2), Q(2,2), S(2)\},$$

then $s_1^1 = I_1$ (denoting versions of s by superscript), and

$$H = P(x,y)P(y,y)Q(y,y)S(y) \rightarrow S(x).$$

Let

$$I_2 = \{P(b,b), Q(a,b), Q(b,b), S(b)\},$$

then

$$\begin{aligned} s^1 \otimes I_2 \quad = \quad & \{P(1b, 2b), P(2b, 2b), Q(2a, 2b), \\ & Q(2b, 2b), S(2b)\}. \end{aligned}$$

If we try to minimise the number of objects in the counter example we have two options, omitting either $1b$ or $2a$. If we omit $2a$ then

$$s_1^2 = \{P(1b, 2b), P(2b, 2b), Q(2b, 2b), S(2b)\}$$

which is isomorphic to s_1^1 so the algorithm makes no progress. If we omit $1b$ then

$$s_1^2 = \{P(2b, 2b), Q(2a, 2b), Q(2b, 2b), S(2b)\}$$

which is a dual case. By using $I_3 = I_1$ we get that $s^2 \otimes I_3$ is isomorphic to $s^1 \otimes I_2$.

6 Unique Substitution Semantics

The above observations indicate that the problem might become easier if a product could not expand a subset of the objects to an interpretation falsifying a new clause. We define an alternative semantics that enforces this restriction, and show that in this setting $\mathcal{F}(P)$ is learnable with a polynomial number of equivalence queries (i.e polynomial in k rather than k^k). The approach we take is similar to the one taken by [Hau89] where (translated to our setting) it is shown that a single universally quantified clause is learnable from equivalence and membership queries (this holds for general clauses not just Horn clauses). The result that follows shows that in this model Horn expressions are also learnable. Thus we extend the result in having more than one clause but restrict the clauses to be Horn.

Definition 6.1 *Let I be an interpretation and $f \in \mathcal{F}(P)$ with variables in X. We say that f is d-true in I (and I is a d-model of f), and denote it by $I \models_d f$ if for all 1-1 substitutions θ that map each variable to a distinct object in I, $f(\theta(X))$ is true in I, where the semantics for ground clauses remains as before.*

Notice that if the number of objects in I is smaller than the number of variables in C then I is positive for C. Caution must be taken with the use of standard inference rules when using this definition because of the shift in semantics (e.g. Modus Ponens must preserve the number of objects in its conclusion). For our purposes it suffices to note that if C_2 can be obtained from C_1 by adding literals to it, and $f \models_d C_1$ then $f \models_d C_2$. Note also that Lemma 3.1 and Lemma 3.2 hold in this model as well. In the rest of the section we discuss only $\models_d$ and omit the special d notation whenever possible. The new semantics define the notion of falsifying a clause. Similarly, we say that I *d-covers* a clause if its antecedent is satisfied in I by a 1-1 substitution that maps all variables of C. Note that this requires that I has enough objects to be mapped to the variables of C. As for falsification we omit d in the rest of the section.

For $\models_d$ McKinsey's theorem does not hold. A product of two models of f may not be a model of f.[1] However, a simpler operation we call *pairing* can be used instead of products and it does not increase the size of the interpretations.

Let I_1, I_2 be interpretations, a *pairing* of I_1, I_2 is an interpretation induced from $I_1 \otimes I_2$ by a subset of the objects that corresponds to a 1-1 matching of the objects in I_1 and I_2. The number of objects in a pairing is equal to the smaller of the number of objects in I_1, I_2. Thus a pairing is not unique and one must specify the matching of objects used to create it. Similar to products we can define k-wise pairings.

Lemma 6.1 *Let $f \in \mathcal{F}(P)$. Then the set of models of f is closed under pairings.*

Proof: Let $f \in \mathcal{F}(P)$, I_1, I_2 models of f, and C a clause in f. Assume that a pairing I falsifies C and consider a substitution $\theta = (\theta_1, \theta_2)$ such that C is falsified by I, where θ_1, θ_2 are the corresponding substitutions mapped to elements of I_1, I_2. Since a pairing is 1-1, both θ_1 and θ_2 are 1-1. We therefore get that the antecedent of C is true in I_1 w.r.t. θ_1, and similarly for I_2, θ_2. Moreover, for at least one of I_1, I_2 the consequent of C is false under the respective substitution. We get that at least one of I_1, I_2 falsifies the clause. ■

[1] For example let $f = P(X,Y) \rightarrow Q(X)$, $I_1 = \{P(1,1)\}$, and $I_2 = \{P(a,b)Q(a)\}$. Both I_1 and I_2 are positive but their product $\{P(1a,1b)\}$ is negative.

Corollary 6.2 *If J is a pairing of I_1 and I_2 and J falsifies $C \in \mathcal{F}(P)$ then at least one of I_1, I_2 falsifies C, and both cover C.*

The following lemma shows that if **A4** is assumed then pairings completely characterise the class $\mathcal{F}(P)^-$. This fact is however not needed for our result that establishes learnability of $\mathcal{F}(P)$.

Lemma 6.3 *Let f be an expression satisfying* **A1, A2, A4**. *If the set of d-models of f is closed under pairings then f can be written as an expression satisfying* **A1, A2, A3, A4** *(i.e. it is Horn).*

Proof: The proof adapts the technique of [McK43] to the current setting. Let $f = \forall X, C_1 \wedge C_2 \wedge \ldots \wedge C_s$ and assume that $C = C_1$ is not Horn, namely it has $j > 1$ positive literals, so that $C = \neg P_1 \vee \ldots \vee \neg P_m \vee P_{m+1} \vee \ldots \vee P_{m+j}$. Define j "Horn-Strengthening" [SK96] clauses for C each including one of the positive literals of C, so that for $1 \leq i \leq j$, $C^i = \neg P_1 \vee \ldots \vee \neg P_m \vee P_{m+i}$.

We claim that for some i, $f \models C^i$ and therefore f can be rewritten as $f = \forall X, C^i \wedge C_2 \wedge \ldots \wedge C_s$. In this way all the non-Horn clauses of f can be replaced with Horn clauses.

To prove the claim assume that for all i, $f \not\models C^i$ and let I_i be be a model of f which is not a model of C^i (which exists since $f \not\models C^i$). Let I be a j-pairing induced from $\otimes(I_1, \ldots, I_j)$ by the objects used in $\overline{\theta} = (\theta_1, \ldots, \theta_j)$ where θ_i is the substitution for I_i which falsifies C^i. For this note that since f satisfies **A4**, all the variables of a clause appear in all versions C^i of that clause and hence all θ_i's have the same variables.

We get that all negative literals of C are true in I (since the component-wise literals must be true in order to falsify the C^i), but for the positive literals at least one of the components is false (e.g. for P_{m+i} the component corresponding to I_i must be false in order to falsify C^i). We therefore get that $I \not\models C$, which contradicts the fact that the models of f are closed under pairings. ■

With pairings used, progress can be monitored similar to the propositional case. The generalised algorithm FOL-Horn-2 is described in Figure 2. The operations $ant(I)$, and $candidates(I)$ are defined as in the relational case in Section 4.2. The algorithm maintains an ordered set S of negative interpretations. These are used to generate the hypothesis by using $candidates(s_i)$ for each $s_i \in S$. On positive counter examples wrong clauses are removed from H. On a negative counter example (as in FOL-Horn-1) the algorithm first greedily minimises the number of objects in the counter example. This ensures that the resulting counter example has at most k objects. The algorithm then tries to find a pairing of this counter example with any of the interpretations s_i in S that results in a negative example which has an extension

1. Maintain an ordered set of interpretations S, initialised to $\emptyset$ and let $H = candidates(S)$.
2. Repeat until $H = f$:
   (a) On a positive counter example remove wrong clauses from H.
   (b) On a negative counter example I:
      i. Minimise the number of objects in I while still negative – (use MQ).
      ii. For $i = 1$ to m (where $S = \{s_1, \ldots, s_m\}$)
         For every pairing J of s_i and I
         if J is negative (use MQ) and it has less objects than s_i or its extension is smaller than that of s_i then
         replace s_i with J, and quit loop.
      iii. If no s_i was replaced then add I as the last element of S.
      iv. Let j be the index of the updated s_i or the added example (i.e. $m + 1$).
         Update H by removing clauses generated by the previous s_j (if a replace)
         and adding the clauses in $candidates(s_j)$ to it.

Figure 2: The algorithm FOL-Horn-2

smaller than that of s_i, or a smaller number of objects. This can be done by trying all possible matchings of objects in the corresponding interpretations and appealing to a membership query oracle.[2] The first s_i for which this happens is replaced with the resulting pairing. In case no such pairing is found for any of the s_i the counter example I is added to S as the last element. Note that the order of elements in S is used in choosing the first s_i to be replaced, and in adding counter example as the last element. These are crucial for the correctness of the algorithm. Finally, note that the algorithm does not need to know the value of k.

The analysis of the algorithm follows the line of argument of [AFP92] establishing that similar properties hold in the more general case. Intuitively, the argument shows that a negative counter example will be "caught" by the first s_i that covers a clause falsified by it. This guarantees that two elements of S do not falsify the same clause of f (since if this happens some previous counter example must not have been caught), and hence yields a bound on the size of S. Since in each step some measurable progress is made, bounds on the number of queries can be derived.

Lemma 6.4 *Let I be a negative counter example after the minimisation of the number of objects (in Step 2(b)i). Assume that the algorithm tests s_i (in Step 2(b)ii). If there is a clause $C \in \mathcal{F}(P)$ such that $f \models C$, s_i covers C, and I falsifies C, then the algorithm replaces s_i.*

Proof: Assume the conditions of the lemma hold. Fix C, and let θ_1 be a substitution showing that s_i covers C, and θ_2 a substitution showing that I falsifies C. Then J, the pairing of the objects that are bound to the same variables in θ_1, θ_2 (this can be done since θ_1, θ_2 are 1-1), falsifies C with respect to $\theta = (\theta_1, \theta_2)$. Therefore J is negative for f.

Since I is negative for C and since its number of objects has been minimised the number of objects in I is exactly the number of variables in C. It follows that either I has less objects than s_i (in which case so do all the pairings and s_i is replaced) in which case we are done, or I and s_i have exactly the same number of objects. Assume therefore that the latter is the case; we argue that the pairing J has a smaller extension than that of s_i. To observe that notice first that a pairing cannot increase the number of positive literals.

Furthermore, consider the clause in $candidates(s_i)$ that corresponds to C and denote it by β. The clause β can be obtained as follows. Since s_i d-covers C there is a 1-1 mapping from objects in s_i to variables in C (this is the inverse of θ_1) so that by following this mapping we can obtain the antecedent of C as a subset of $ant(s_i)$. To get β, assume this mapping of variables is used, and pick the element of $candidates(s_i)$ that has the same consequent as C. There are two cases: if the consequent of C is already in $ant(s_i)$ then β is trivially true (it has the consequent as part of the antecedent). This happens if s_i d-covers but does not d-falsify C. In this case we may assume that β is in H. In the other case, when the consequent of C is not in $ant(s_i)$, β is indeed in $candidates(s_i)$. The next argument holds in both cases.

Since β can be obtained from C by adding literals to it we have that $f \models \beta$ and therefore it is not removed from H by any positive counter example. We also know that since I is a negative counter example it is positive for β, and in particular β is not falsified by I with respect to θ_2.[3] Now since I falsifies C under θ_2 it must be the case that the consequent of C is false in I under θ_2 and since the consequent is the same in C and β the same holds for β. We therefore get that the antecedent of β is not true in I with respect θ_2, or in other words there is a literal $l(X)$ in β such that $l(\theta_2(X))$ is false in I. The literal $l(X)$ was generated by $l(\theta_1(X))$ in s_i. Since the pairing J matches objects according to the variables they are bound to we get that, while $l(\theta_1(X))$ is in s_i, $l(\theta(X))$ is not in J where $\theta = (\theta_1, \theta_2)$, and thus J has a smaller extension. ■

Lemma 6.5 *At all times in the algorithm, for all k, i such that $k < i$, and for all $C \in \mathcal{F}(P)$ such that $f \models C$, if s_i falsifies C then s_k does not d-cover C.*

[2] Obviously a better solution for this would be desirable. One can show that certain greedy methods will not succeed.

[3] Notice that since C and β have the same variables θ_2 can be used for both, and thus the above is well defined.

Proof: We argue by induction on the construction of S. The claim clearly holds for the empty set. For the inductive step, assume the claim does not hold; we show that a contradiction arises. Let I be the last counter example, and let C be the clause that exists if the claim does not hold.

Consider first the case where $I = s_i$ is appended. But in this case I falsifies C, and by Lemma 6.4 s_k is replaced if tested.

Clearly we only need to argue about cases where either s_i or s_k are replaced. Consider next the case where s_k is replaced by J. By Corollary 6.2, since J falsifies C, s_k d-covers C, and this contradicts the inductive assumption.

Consider next the case where s_i is replaced by J. Then J is negative and therefore (again by Corollary 6.2) both s_i and the counter example I, d-cover C, and at least one falsifies C. If s_i falsifies C we get a contradiction to the inductive assumption. If I falsifies C then by Lemma 6.4 s_k is replaced if tested. ■

Theorem 6.6 *The class $\mathcal{F}(P)$ is learnable by the algorithm FOL-Horn-2 using equivalence queries and membership queries under the unique substitution semantics. For $f \in \mathcal{F}^k(P)$ with m clauses, the algorithm makes at most $E_N + E_P$ equivalence queries and $(n+mk^k)E_N$ membership queries, where $E_N \leq m(\alpha+k)$, $E_P \leq E_N\alpha$, n is the largest number of objects in any of the counter examples, and $\alpha = |P|k^a$ where a is the bound on arity of predicates. The running time of the algorithm is polynomial in the above bounds and n^k.*

Proof: Since all elements of S are negative, each falsifies at least one clause of f. By Lemma 6.5, no two elements falsify the same clause of f and hence at any time S has at most m elements.

Every negative counter example either reduces the number of objects in some s_i, or reduces the size of the extension of some s_i, or introduces a new element s_i. The size of the extension of any I with at most k objects is bounded by $|P|k^a$, and each s_i has at least one and at most k objects. The number of negative counter examples E_N is therefore bounded by $m(\alpha + k)$.

After each negative counter example the algorithm updates H by changing the clauses of a single s_j. Since the number of possible consequents is bounded by α this produces at most α wrong clauses. Since every positive counter example removes at least one wrong clause from H, there are at most $E_N\alpha$ positive counter examples. This derives the bound on the number of equivalence queries.

For the membership queries notice that for each negative counter example we need at most n queries for reducing the number of objects, and at most mk^k queries to test pairings.

Considering the running time, the operations on negative examples are polynomial in the above bounds. For a positive counter example I the algorithm has to evaluate each clause in H on I, and this can be done in time $O(n^k)$, since clauses in H have at most k variables. ■

By careful recording we can make sure that each consequent in $candidates(s_i)$ is removed only once and in this way reduce the number of positive counter examples. This can be done since if $AB \rightarrow C$ is not implied by f (and it is removed), then clearly $A \rightarrow C$ is not implied by f. It can be seen that for a fixed i, the antecedents of clauses in $candidates(s_i)$ are subsets of previous antecedents. Hence once a consequent is removed for s_i, as a result of a positive counter example, it need not be generated again when updating H. Hence E_P can be reduced to $m\alpha$. This idea is discussed in detail for the propositional case in [AFP92].

7 Extensions

In this section we apply Theorem 6.6 to other settings. In doing so we omit the exact bounds which can be easily derived. For a related discussion and comparison of various models of learning when queries are not allowed see [DR97].

7.1 Normal Semantics

We can apply the theorem to the normal semantics since expressions in $\mathcal{F}^k(P)$ under $\models_d$ can simulate expressions in $\mathcal{F}^k(P)$ under $\models$.

Lemma 7.1 *For every $f \in \mathcal{F}^k(P)$ with m clauses there is an expression $U(f) \in \mathcal{F}^k(P)$ with at most mk^k clauses such that for all interpretations I, $I \models f$ if and only if $I \models_d U(f)$.*

Proof: We construct $U(f)$ from f by considering every clause separately. For a clause C in f with j variables generate set of clauses $U(C)$. To do that, consider all partitions of the j variables; each such partition generates a clause by assigning a single new variable to all variables in a single class. This covers all possibilities of unifying various subsets of variables of C to each other. The number of such clauses is equal to the number of partitions of a j element set (the Bell number B_j) that is obviously bounded by j^j. Then $U(f)$ is the conjunction of all clauses generated for all its clauses. The construction makes sure that all possible ways to falsify a clause C in f by a non-unique substitution are covered by a unique substitution for one of the clauses of $U(C)$. It is easy to check that the claim follows. ■

Hence the algorithm working under $\models_d$ can simply interact with oracles working according to $\models$ and still learn the same class.

Corollary 7.2 *The class $\mathcal{F}(P)$ is learnable from equivalence and membership queries.*

The above corollary improves upon Theorem 4.1 by extending the concept class. It does result however in an expansion similar to the propositional one, though slightly better. While the propositional expansion has an element of S for each possible substitution for each clause, the current expansion has an element for each partition of its variables. Similar differences hold for the bounds on the number of equivalence queries.

7.2 Using Equality

As we now show another advantage of $\models_d$ is that it allows for an easy incorporation of equalities and inequalities relative to $\models$. In particular $\mathcal{F}^k(P)$ under $\models_d$ can simulate $\mathcal{F}^k(P,=)$ under $\models$ hence yielding a learning result for $\mathcal{F}(P,=)$ under $\models$.

Lemma 7.3 *For every $f \in \mathcal{F}^k(P,=)$ with m clauses there is an expression $U^*(f) \in \mathcal{F}^k(P)$ with at most mk^k clauses such that for all interpretations I, $I \models f$ if and only if $I \models_d U^*(f)$.*

Proof: We first show that for each $f \in \mathcal{F}(P,=)$ there is an expression g equivalent to f under $\models$ such that no clause in g includes inequalities; g has the same number of clauses as f. (Hence inequalities are in some sense useless.) To get g from f consider each clause separately. For each inequality $(x_i \neq x_j)$ in a clause C replace all occurrence of x_i and x_j in C by $x_{\min\{i,j\}}$ and remove the inequality from C. Repeat this until there are no more inequalities in C.

Now, if $I \not\models f$ then there is a substitution θ and a clause C that is falsified by it. Consider any inequality in C. Since the inequality is not satisfied, θ maps both its variables to the same object. Hence all the variables of C re-mapped to a single variable in g are mapped to a single object by θ. Since we kept one of these variables as the representative, all the literals in the corresponding clause of g have the same value under θ an hence it is falsified by I.

On the other hand if $I \not\models C'$ for a clause C' in g then one can extend the substitution to cover variables of the corresponding clause C in f by mapping all the variables unified in the generation of C' to the same object. Clearly, I falsifies all literals that are retained in C' under this substitution. By construction it also falsifies the inequalities, and hence falsifies C.

We next construct $U^*(f)$ from g by considering every clause separately. For a clause C in g generate set of clauses $U^*(C)$. Consider a clause C in g and the clauses $U(C)$ as generated in Lemma 7.1 ignoring equalities and inequalities. Now consider a positive literal $(x_i = x_j)$ in the clause C. The clause is satisfied under any substitution in which x_i and x_j are mapped to the same object. Hence we can remove from $U(C)$ all those clauses where x_i and x_j were mapped to the same variable. This can be repeated for all equalities in C to generate $U^*(C)$. The conjunction of all clauses in $U^*(C)$ for all C in g constitutes $U^*(f)$.

Assume $I \not\models g$ for some I. Thus some clause C in g is falsified by I under some substitution θ. Partition the variables of C according to the objects they are mapped to in θ, generating a clause from this partition as in the generation of $U(C)$. We claim that the resulting clause has not been removed from $U^*(C)$. This is true since all equalities in C are not satisfied and thus their variables are mapped to distinct objects. Hence this clause is falsified by I under the same substitution.

Finally, assume $I \not\models_d U^*(f)$ for some I. Thus some clause C' in $U^*(f)$ is falsified by I under some substitution θ mapping distinct variables to distinct objects. Let C be the clause that generated C', and extend θ to variables of C by using the inverse mapping of the variable partition used when generating C' from C. We claim that I falsifies C under the extended substitution. For this first observe that all literals in C not involving equality are falsified since they have the same values as in C' under θ. Consider next an equality $(x_i = x_j)$ in C. All elements of $U(C)$ in which x_i and x_j are mapped to the same variable have been removed from $U^*(C)$. It follows that x_i and x_j are mapped to different variables in C' and since θ maps each variable of C' to a unique object the equality is falsified. Hence all literals of C are falsified, and $I \not\models g$. ■

We therefore get:

Corollary 7.4 *The class $\mathcal{F}(P,=)$ is learnable from equivalence and membership queries.*

Notice that in the above result the hypothesis of the algorithm can be converted into an expression in $\mathcal{F}(P,=)$ with respect to $\models$. This can be done by adding equalities on all the variables in all clauses. That is $p(x,y)p(y,z) \rightarrow q(z)$ (under $\models_d$) will be translated to $p(x,y)p(y,z) \rightarrow q(z) \vee (x = y) \vee (x = z) \vee (y = z)$ or equivalently to $p(x,y)p(y,z)(x \neq y)(x \neq z)(y \neq z) \rightarrow q(z)$.

7.3 Entailment Queries

Another corollary can be derived showing that entailment oracles are also sufficient for learning. We first observe that membership queries can be replaced with entailment queries. Let $candidates(I)$ be the propositional operation of Section 4.1.

Lemma 7.5 *Let I be an interpretation and $f \in \mathcal{F}(P)$. Then $I \not\models f$ if and only if for some $c \in candidates(I)$, $f \models c$.*

Proof: Clearly, for all $c \in candidates(I)$ the antecedent of c is satisfied by I and its consequent is not, and therefore $I \not\models c$. Hence, if $f \models c$, then $I \not\models f$.

For the other direction assume $I \not\models f$. Therefore it falsifies some clause C of f under some substitution θ. Consider $c' = C(\theta(X))$ the ground instance of C obtained by following θ. Clearly, $I \not\models c'$, and therefore its antecedent is true in I (and therefore is a subset of the extension of I), and its consequent is not. Now consider the clause c whose antecedent is $ant(I)$ and whose consequent is identical to the consequent of c'. Then c is in $candidates(I)$ and $f \models C \models c' \models c$. ■

Therefore when the algorithm presents a membership query we can ask a sequence of entailment queries and answer no if and only if one of them is implied by f. Moreover, if entailment queries are available we can make sure when creating a hypothesis that its clauses are always implied by f. This can be done by asking an entailment query for each of the clauses in $candidates(s_i)$. Since the relational version of $candidates()$ is used by the algorithm these clauses must be translated to ground clauses. This can be done by substituting an arbitrary distinct object for every variable in the clause. We therefore reduce the n^k dependence in the running time (needed for positive counter examples). We get:

Corollary 7.6 *The class $\mathcal{F}(P)$ is learnable from equivalence queries and entailment queries.*

7.4 ILP

A similar application can be obtained for ILP. We consider the setting as defined in [Coh95b]. In this setting, background knowledge B is given to the learner and a Horn expression f is to be learned. It is assumed that the background knowledge B is a conjunction of positive ground literals.

An example is meant as a positive example for some concept in the world (which is the consequent in some clause in f). In particular an example is a pair (E, D) such that D (for Description) is a conjunction of positive ground literals and E is a single positive ground literal. An example (E, D) is a positive example for f with respect to B if and only if $f \wedge B \wedge D \models E$ or alternatively $f \wedge B \models (D \rightarrow E)$. Thus we see that a positive ILP example is the same as an "entailment example". Let a *ILP equivalence oracle* be such that when presented with a hypothesis H, it answers yes if and only if $f \wedge B$ is equivalent to $H \wedge B$ (under $\models$), and otherwise it returns a ILP counter example, namely, a pair (E, D) which is positive for one of f or H but not the other. The following lemmas show that, when an entailment oracle is available (and $f \models H$), a ILP equivalence oracle can be used instead of the standard equivalence oracle. For the results to apply we need one further restriction, known as definite clauses:
A3': All clauses have *exactly* one positive literal.

Let $\mathcal{F}_D(P)$ be the class of expressions satisfying the assumptions **A1, A2, A3'**.

Lemma 7.7 *If $f \models H$ and (E, D) is a counter example for H then it is the case that $f \wedge B \models (D \rightarrow E)$ and $H \wedge B \not\models (D \rightarrow E)$.*

Proof: If $H \wedge B \models (D \rightarrow E)$ then $f \wedge B \models H \wedge B \models (D \rightarrow E)$. ■

Lemma 7.8 *Let $f, H \in \mathcal{F}_D(P)$. If $f \models H$ and (E, D) is a counter example for H then an interpretation I such that $I \not\models f$ and $I \models H$ can be found in time $O(|P|n^k n^a)$ where n is the number of objects in E, D, and B.*

Proof: Let I be an interpretation that includes the objects in B, D, and E, and whose extension includes precisely the positive literals in B and D. The idea [FP93, RT98] is to compute the "closure" of I with respect to H. Clearly, $I \not\models f$. If $I \models H$ then we are done. Otherwise, we can find a clause C of H falsified by I under a substitution θ. Let the ground consequent of $C(\theta(X))$ be γ; add γ to I and repeat the process until $I \models H$. This must eventually happen since as a result of assumption **A3'** H is satisfied by the interpretation that has all ground literals true. We claim that I satisfies the conditions of the lemma.

Since $H \not\models (B \wedge D \rightarrow E)$, E is never added to I and hence $I \not\models f$. On the other hand, by the construction $I \models H$. It only remains to observe that the number of iterations is bounded by the maximum size of an interpretation with n objects. ■

Entailment queries are also natural in this setting since they are in some sense "ILP membership queries". A *ILP entailment oracle* for B and f when presented with a ground clause c answers yes if and only if $f \wedge B \models c$. The only problem one needs to get around in order to apply Corollary 7.6 is the fact that B is part of the problem specification and thus we need to make sure that B does not effect the answers to our queries. This can be done by using distinct new object names in the queries (that do not appear in B). Since, by using an entailment oracle we can guarantee that $f \models H$, we get:

Corollary 7.9 *The class $\mathcal{F}_D(P)$ is learnable from ILP equivalence queries and ILP entailment queries.*

It seems plausible that the above result can be extended to deal with non-ground background knowledge $B \in \mathcal{F}_D(P)$ by incorporating B into the hypothesis of the algorithm.

8 Concluding Remarks

We have shown that universally quantified function free Horn expressions are learnable in several models of exact learning from queries. The most expressive class shown learnable allows for an arbitrary number of equalities to appear in the expressions thus going slightly beyond pure Horn expressions. Our algorithms are superpolynomial in the number of universally quantified variables though polynomial in other parameters.

Various questions remain to be resolved, like, allowing constants and function symbols in the learned expressions, improving the complexity or proving lower bounds, and allowing for alternation of quantifiers. Another question regards the use of these ideas in a practical ILP system say for learning domain knowledge (from an expert). One difficulty no doubt will be the number of membership queries that need to be answered. Some ideas on how to reduce the number of queries in practice appear in [Sha83].

We next briefly mention two likely extensions of the results. In our proof for the theorem instead of arguing about clauses in the representation of f we argued on Horn clauses implied by f. It may be possible to utilise this to show that least Horn upper bounds[4] [SK96, KR96, KR97, FP93] are learnable from random examples and entailment queries. This might also have some implications for learning to reason [KR97] where these constructs are used.

Our framework raises the question of how to treat constants in the learning protocol. In most ILP frameworks, object names and constants are treated in the same way (e.g. [RT97]), the constants are those that remain after the generalisation process. When using interpretations on the other hand the standard logical definitions [CK90, Llo87] suggest that constants' names are fixed, and are "bound" to objects in the interpretation. This makes its application somewhat awkward and less natural. It seems that the algorithm Prop-Horn-1 can be adapted to deal with this case, adding a polynomial factor in the number of constants in the language.

Acknowledgments

I am grateful to Dan Roth for many discussions regarding the notion of products and the results in [McK43], and to Mark Jerrum, Chandra Reddy, and Prasad Tadepalli for comments on earlier drafts.

[4] For any expression f, the least Horn upper bound of f is an expression equivalent to the conjunction of all Horn clauses implied by f. Such an expression can be used to to decide whether $f \models \alpha$ for any Horn expression α.

Appendix

A Proof of Theorem 4.1

We first show that if entailment queries are also allowed then the algorithm can be used to learn the class $\mathcal{F}(P)^-$. For this we modify the algorithm as follows. The set $clauses(I)$ is the set of clauses $\{C \in candidates(I) \mid \text{such that } f \models C\}$. Given a set S of interpretations, $clauses(S)$ can be computed by appealing to an entailment oracle. Notice that since $s_i \not\models f$, s_i falsifies at least one of the clauses of f and by virtue of that and the construction, we get that $clauses(s_i)$ is not empty. The hypothesis of the algorithm is now computed by $H = \wedge_{s_i \in S} clauses(s_i)$.

Lemma A.1 *The class $\mathcal{F}(P)^-$ is learnable by the modified FOL-Horn-1 using equivalence queries, membership queries, and entailment queries. For $f \in \mathcal{F}^k(P)^-$ with m clauses, the algorithm is polynomial in $m, |P|, k^a, k^k, n$, where n is the largest number of objects in the counter examples.*

Proof: Note that by the use of the entailment oracle we are guaranteed that at all times $f \models H$, and therefore a counter example is such that $I \not\models f$ and $I \models H$.

Lemma 3.3 identifies a target expression f_p for the learning problem for Prop-Horn. The correctness and complexity bound follow from those of Prop-Horn if we can show that the simulation is correct. It suffices to show that (1) the membership queries are answered correctly according to f_p, (2) if Prop-Horn asks an equivalence query and if $H \neq f$ then the algorithm will present a counter example x to Prop-Horn, and (3) x is indeed a counter example for the internal hypothesis of Prop-Horn and the target function f_p. Part (1) follows immediately by Lemma 3.3.

For (2) note that if $H \neq f$ then a counter example for H is returned, and some x is passed to Prop-Horn. Note also that as argued above the reduced interpretation I is a counter example for H and using Lemma 3.3 again we get that x is a counter example for H.

For (3), we claim that h the internal hypothesis of Prop-Horn is satisfied by x, and thus x is a counter example.[5] Assume x falsifies h. Then one of the clauses c in h is falsified. Let s_i be the interpretation that generated c and let C be the corresponding clause of H. Clearly, there is a substitution θ, the inverse of the one used for the generation of C, so that C is falsified by x, contradicting the fact that x is a counter example for H.

The generation of H can be done in time polynomial in $|S|$ (since the arity is bounded and interpretations are of polynomial size). The minimisation of the counter examples requires a number of queries linear in the number of objects in the original counter example. For the rest the complexity is governed by the complexity of Prop-Horn which is polynomial in the number of propositional variables and the size of f_p. The latter is $O(mk^k)$ where $f \in \mathcal{F}^k(P)^-$ has m clauses.

[5] Here we only consider clauses c such that $f_p \models c$, and therefore have $f_p \models h$. Since the internal hypothesis is never created we may assume a modified version of Prop-Horn that appeals to an entailment oracle and includes only correct clauses in its hypothesis. This modified version is obviously correct and suffices for the current argument.

Lastly, consider the case where k is not known. We start with $k = 1$ and run as before unless we find that a counter example cannot be minimised to have k objects. We then increase k to the next possible size. Correctness follows since as long as we do not meet counter examples that are too large, the propositional learning problem simulates the learning of f when restricted to interpretations of size k. (Essentially the construction of f_p can be generalised to have i^k substitutions when considering i objects.) We therefore have at most k iterations (and can use doubling to reduce the number of iterations to $\log k$), where in each iteration the complexity is bounded as before. ■

Finally, to prove the theorem we show that entailment queries are not needed.

Proof: **(Theorem 4.1)** The modified FOL-Horn-1 algorithm uses Prop-Horn as a black box. The role of Prop-Horn is however reduced to manipulating the set S. Namely, the hypothesis need not be generated. The manipulation of S consists of computing the intersection of two interpretations and in asking membership queries to decide on the update.

In the previous lemma entailment queries were used to ensure that we always get negative counter examples. When using the hypothesis $H = \wedge_{s_i \in S} candidates(s_i)$, the algorithm may get positive counter examples, that are used to removed wrong clauses from H. This can be done in time $O(n^k)$ for each clause in H. Since the number of wrong clauses in H is bounded by the size of the sets $candidates(s_i)$ the same bounds follow (essentially the entailment queries are traded for positive counter examples). ■

References

[AFP92] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. *Machine Learning*, 9:147–164, 1992.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[CK90] C. Chang and J. Keisler. *Model Theory*. Amsterdam Oxford : North-Holland, 1990.

[Coh95a] W. Cohen. PAC-learning recursive logic programs: Efficient algorithms. *Journal of Artificial Intelligence Research*, 2:501–539, 1995.

[Coh95b] W. Cohen. PAC-learning recursive logic programs: Negative result. *Journal of Artificial Intelligence Research*, 2:541–573, 1995.

[DMR92] S. Dzeroski, S. Muggleton, and S. Russell. PAC-learnability of determinate logic programs. In *Proceedings of the Conference on Computational Learning Theory*, pages 128–135, Pittsburgh, PA, 1992. ACM Press.

[DR97] L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95(1):187–201, 1997.

[DRB92] L. De Raedt and Bruynooghe. An overview of the interactive concept learner and theory revisor CLINT. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.

[DRD94] L. De Raedt and S. Dzeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

[FP93] M. Frazier and L. Pitt. Learning from entailment: An application to propositional Horn sentences. In *Proceedings of the International Conference on Machine Learning*. Morgan Kaufmann, 1993.

[Hau89] D. Haussler. Learning conjunctive concepts in structural domains. *Machine Learning*, 4(1):7–40, 1989.

[Hor51] A. Horn. On sentences which are true on direct unions of algebras. *Journal of Symbolic Logic*, 16(1):14–21, 1951.

[HST97] T. Horvath, B. Sloan, and G. Turan. Learning logic programs by using the product homomorphism method. In *Proceedings of the Conference on Computational Learning Theory*, pages 10–20, 1997.

[HT95] T. Horvath and G. Turan. Learning logic programs with structured background knowledge. In *The fifth ILP workshop*, 1995.

[Kha96] R. Khardon. Learning to take actions. In *Proceedings of the National Conference on Artificial Intelligence*, pages 787–792, Portland, Oregon, 1996. AAAI Press.

[KR96] R. Khardon and D. Roth. Reasoning with models. *Artificial Intelligence*, 87(1-2):187–213, 1996.

[KR97] R. Khardon and D. Roth. Learning to reason. *Journal of the ACM*, 44(5):697–725, 1997.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1987. Second Edition.

[McK43] J. C. C McKinsey. The decision problem for some classes of sentences without quantifiers. *Journal of Symbolic Logic*, 8(3):61–76, 1943.

[MDR94] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 20:629–679, 1994.

[Plo70] G. D. Plotkin. A note on inductive generalization. In *Machine Intelligence 5*, pages 153–163. American Elsevier, 1970.

[PY97] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. In *Proceedings of the symposium on Principles of Database Systems*, pages 12–19, 1997.

[RT97] C. Reddy and P. Tadepalli. Learning Horn definitions with equivalence and membership queries. In *The seventh ILP workshop*, 1997.

[RT98] C. Reddy and P. Tadepalli. Learning first order acyclic Horn programs from entailment. In *International Conference on Machine Learning*, 1998. Forthcoming.

[RTR96] C. Reddy, P. Tadepalli, and S. Roncagliolo. Theory guided empirical speedup learning of goal decomposition rules. In *International Conference on Machine Learning*, pages 409–416, 1996.

[Sha83] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.

[SK96] B. Selman and H. Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, March 1996.

[Val84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[Val85] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pages 560–566, Los Angeles, CA, 1985. Morgan Kaufmann.

[Val94] L. G. Valiant. *Circuits of the Mind*. Oxford University Press, Oxford, UK, November 1994.