Learning Function-Free Horn Expressions

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

September 22, 1998

Abstract

The problem of learning universally quantified function free first order Horn expressions is studied. Several models of learning from equivalence and membership queries are considered, including the model where interpretations are examples (Learning from Interpretations), the model where clauses are examples (Learning from Entailment), models where extentional or intentional background knowledge is given to the learner (as done in Inductive Logic Programming), and the model where the reasoning performance of the learner rather than identification is of interest (Learning to Reason). We present learning algorithms for all these tasks for the class of universally quantified function free Horn expressions. The algorithms are polynomial in the number of predicate symbols in the language and the number of clauses in the target Horn expression but exponential in the arity of predicates and the number of universally quantified variables. We also provide lower bounds for these tasks by way of characterising the VC-dimension of this class of expressions. This shows that apart for the dependency on the number of variables our algorithms are essentially optimal.

1 Introduction

We study the problem of exactly identifying first order Horn expressions using Angluin's (1988) 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 were given by Valiant (1985) and Haussler (1989), but only recently more attention was given to the subject in framework of Inductive Logic Programming (see e.g. Muggleton

& De Raedt, 1994; Cohen, 1995a, 1995b). 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 (Cohen, 1995b). To tackle this issue in the propositional domain various queries and oracles that allow for efficient learning have been studied (Valiant, 1984; Angluin, 1988). In particular, propositional Horn expressions are known to be learnable in polynomial time from equivalence and membership queries (Angluin, Frazier, & Pitt, 1992), and from entailment queries (Frazier & Pitt, 1993). In the relational domain, queries have been used in several systems (e.g. Shapiro, 1983; Sammut & Banerji, 1986; De Raedt & Bruynooghe, 1992; Muggleton & Buntine, 1992) and results on learnability in the limit were derived for some (e.g. Shapiro, 1983; De Raedt & Bruynooghe, 1992). More recently Reddy and Tadepalli (1997, 1998) considered the use of equivalence and membership queries and have shown that Horn definitions (where all clauses have the same unique positive literal), and acyclic Horn expressions are learnable.

In this paper we show that function free universally quantified Horn expressions are exactly learnable in several models of learning from equivalence and membership queries. One distinction between the learning models is concerned with the notion of examples. The natural generalisation of the setup studied in propositional logic suggests that examples are interpretations of the underlying language. That is, a positive example is a model of the expression being learned. Another view suggests that a positive example is a sentence that is logically implied by the expression, and in particular Horn clauses have been used as examples. These two views have been called *learning from interpretations* and *learning from entailment* respectively (De Raedt, 1997) and were both studied before.

Another aspect of the learning models is the use of background knowledge in the process of learning. This idea has been formalised in *Inductive Logic Programming* (ILP) where the background knowledge is given to the learner as a logical expression in the same language as that of the target expression being learned (Muggleton & De Raedt, 1994). The background knowledge may be *extensional* that is a set of ground facts, or *intentional* where it may include arbitrary expressions in the language. Finally, the framework of *Learning to Reason* has been suggested for the study of systems that learn their knowledge in order to reason with it about the domain in question. Instead of finding an expression equivalent to the domain description, the learner is expected to learn some representation with which it can perform the reasoning correctly as long as the reasoning queries of interest belong to a restricted class of expressions.

We present algorithms for all these tasks with respect to universally quantified function free Horn expressions. Our method follows closely the results from the propositional domain (Angluin et al., 1992; Frazier & Pitt, 1993) generalising these by finding appropriate first order constructs. Thus one contribution of the paper is in lifting the results to the first order domain and developing the appropriate algorithms. Another contribution is in developing techniques for converting learning algorithms from one model to another, thus clarifying the some of the relationships between the various models. Finally, we characterise the VC-dimension of the class under consideration; this is a combinatorial parameter known to provide a lower bound for the complexity of learning (Blumer, Ehrenfeucht, Haussler, & Warmuth, 1989; Maass & Turán, 1992). The result shows that in several of the parameters the complexity of our algorithm is essentially optimal.

To illustrate the complexity of the algorithms consider the Horn expression (exact definitions for the various notions appear in the next sections)

$$\begin{aligned}&\forall x_1, x_2, x_3,\\ &(p_1(x_1,x_2)p_2(x_1,x_3) \rightarrow p_1(x_2,x_1)) \wedge\\ &(p_3(x_3,x_1)p_1(x_3,x_1) \rightarrow p_4(x_3)).\end{aligned}$$

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 size (number of objects) in the examples it sees, and time complexity polynomial in the above parameters and n^k. The lower bound for the number of queries following from the VC-dimension is $\Omega(m|P|k^a)$.

Our results are derived by first considering the case of learning from interpretations. We describe two slightly different algorithms (with different proofs of correctness) that perform this task. Algorithms for other tasks are then constructed from the solution of this task. One of the main results of the paper is concerned with deriving one of the basic algorithms. In doing that we use a variant of the standard semantics where each universally quantified variable in an expression must be bound to a unique element. We show that in this setting the number of equivalence queries is polynomial in k (rather than k^k) whereas the running time and membership queries are as above. Thus in this model the number of equivalence queries the algorithm makes is essentially optimal. Our learning algorithm uses *pairings* of examples, an operation that is a variant of direct products (that have been used before for learning (Horvath, Sloan, & Turán, 1997)). A similar modified semantics has been considered before by Haussler (1989). 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. Another interesting aspect of the modified semantics is that it can be used to derive a learning algorithm for a more expressive language (under the normal semantics) 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.

While this paper concentrates on function free expressions, extending the results to more expressive languages in clearly of interest. One such result for range restricted Horn expressions (where the way function symbols are used in clauses is restricted) has been recently developed using a reduction to the function free case (Khardon, 1998).

The rest of the paper is organised as follows. Section 2 gives preliminary definitions and details. Section 3 presents some simple examples that motivate the construction developed in Section 4 where the result on learning with the special semantics is proved. Section 5 extends this result for other learning models, and Section 6 characterises the VC dimension of the class. Section 7 develops the second basic algorithm showing how the propositional algorithm can be used more directly in the first order domain. Finally Section 8 concludes 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 assume a pre-fixed known and finite signature of the language. Constants or other function symbols are not allowed in the language. That is, the signature is a finite set of predicates P each with its associated arity. In addition a set of variables $x_1, x_2, x_3, \ldots$ is used to construct expressions.

Definitions of first order languages can be found in standard texts (e.g. Chang & Keisler, 1990; Lloyd, 1987); 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; a Horn clause is said to be *definite* if it has precisely one positive literal. A Horn expression is a conjunction of Horn clauses. Note that any clause can be written as $c = (\wedge_{n \in \mathrm{Neg}} n) \rightarrow (\vee_{p \in \mathrm{Pos}} p)$ where Neg and Pos are the sets of negative and positive literals of C respectively. When doing so we will refer to $(\wedge_{n \in \mathrm{Neg}} n)$ as the *antecedent* of c and to $(\vee_{p \in \mathrm{Pos}} p)$ as the *consequent* of c.

Definition 1 *Let $C = (\wedge_{n \in Neg} n) \rightarrow (\vee_{p \in Pos} p)$ be a clause, then C is* range restricted[1] *if every variable that appears in Pos also appears in Neg.*

Note that since the expressions of interest are function free, range restricted clauses have non-empty antecedents.

Definition 2 *Let $\mathcal{H}(P)$ be the set of (function free) Horn expressions over signature P, $\mathcal{H}(P)^-$ the set of expressions in $\mathcal{H}(P)$ in which all clauses are range restricted, and $\mathcal{H}_D(P)^-$ the set of expressions in $\mathcal{H}(P)^-$ in which all clauses are definite.*

For example, $(p(x,y) \rightarrow q(x) \vee q(y))$ is not Horn and not in any of the classes above although it is range restricted, $(p(x,y) \rightarrow q(z))$ and $q(x)$ are in $\mathcal{H}(P)$ but not in $\mathcal{H}(P)^-$, $(\overline{p(x,y)} \vee \overline{p(y,z)})$ is in $\mathcal{H}(P)^-$ but not in $\mathcal{H}_D(P)^-$, and $(p(x,y) \rightarrow q(x))$ is in $\mathcal{H}_D(P)^-$.

[1] This restriction has been used before by several authors. Unfortunately, in a previous version of this paper it was called "non-generative" while in other work it was called "generative" (Muggleton & Feng, 1992). The term "range-restricted" was used in database literature (see e.g. Minker, 1988).

Definition 3 *Let $\mathcal{H}(P,=)$ be the language $\mathcal{H}(P)$ 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 that appear in relational literals in the clause.*

The class $\mathcal{H}(P,=)$ goes somewhat beyond Horn expressions if equalities are considered as positive literals. An example clause in $\mathcal{H}(P,=)$ appears in the introduction.

2.2 Examples

An example is an interpretation I of the predicates in P (Lloyd, 1987). 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 *size* of an interpretation is the sum of sizes of extensions of predicates in it. If the arity of all predicates is bounded by a constant a then the size of the extension of an example is polynomial in the number of domain elements.

Examples of this form have been used by Haussler (1989) and are motivated by the scenario of acting in structural domains (e.g. Khardon, 1996; Reddy, Tadepalli, & Roncagliolo, 1996; Reddy & Tadepalli, 1997). They are also used in the non-monotonic form of ILP (De Raedt & Dzeroski, 1994). 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, 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 (Chang & Keisler, 1990; Lloyd, 1987).

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 object. A ground negative literal is true in I if and only if its negation is not.

A clause $C \in \mathcal{H}(P)$ 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 $T \in \mathcal{H}(P)$ is true in I if all clauses C in T are true in I. The terms (1) T is true in I, (2) I is a positive example for T, (3) I satisfies T, (4) I is a model of T, and (5) $I \models T$, have the same meaning. Let $T_1, T_2 \in \mathcal{H}(P)$ then T_1 implies T_2, denoted $T_1 \models T_2$, if every model of T_1 is also a model of T_2.

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 $\mathcal{H}^k(P), \mathcal{H}^k(P)^-, \mathcal{H}^k_D(P)^-$, and $\mathcal{H}^k(P,=)$.

For $T \in \mathcal{H}^k(P,=)$, one can test whether $I \models T$ 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 if k is not bounded, and recent results suggest that it is not likely to have an algorithm polynomial in n even for small non-constant values of k (Papadimitriou & Yannakakis, 1997). 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 (1989).

Other assumptions that ensure tractability, e.g. determinacy (Dzeroski, Muggleton, & Russell, 1992), have been used before but we do not address such restrictions here.

2.5 The Learning Model

The learning model uses several forms of queries (Angluin, 1988; Frazier & Pitt, 1993). Let $\mathcal{H}$ be a class under consideration, and let $T \in \mathcal{H}$ be the target expression. For *Equivalence Queries* (EQ) the learner presents a hypothesis $H \in \mathcal{H}$ and the oracle returns "yes" if $H = T$ and otherwise it returns an interpretation I that is a counter example ($I \models T$ and $I \not\models H$ or vise versa). For *Membership Queries* (MQ) the learner presents an interpretation I and the oracle returns "yes" iff $I \models T$. In the learning model a target expression $T \in \mathcal{H}$ is fixed and hidden from the learner. A learner interacts with the oracles and has to find an expression H that is equivalent to T with respect to $\models$.

We also study oracles based on entailment where clauses serve as examples. Intuitively, for Entailment Membership Queries, the learner presents $C(\theta(X))$, a ground instance of a clause $C \in \mathcal{H}$ (i.e. all variable are substituted to objects) and the oracle returns "yes" iff $T \models C(\theta(X))$. The above is however not well defined since the ground clause is not in the language $\mathcal{H}(P)$ and the models of T do not have an interpretation for the constants in C. Precise definitions for entailment oracles as well as the ILP setting are given in Section 5.

2.6 Small Interpretations

The following lemmas indicate that we may 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

Table 1: The algorithm Prop-Horn: learn propositional Horn expressions using EQ and MQ

```
1. Maintain an ordered set of interpretations $S$, initialised to $\emptyset$ and let $H = \textit{prop-cands}(S)$.
2. Repeat until $H = T$:
   • Ask an equivalence query to get a counter example in case $H \neq T$.
   • On a positive counter example $I$ (s.t. $I \models T$) remove wrong clauses (s.t. $I \not\models C$) from $H$.
   • On a negative counter example $I$ (s.t. $I \not\models T$):
       For $i = 1$ to $m$ (where $S = (s_1, \ldots, s_m)$)
           If $J = s_i \wedge I$ is negative (use MQ),
              and its size 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 $\textit{prop-cands}(S)$.
```

predicates on these objects. Let $\mathcal{I}^k$ be the set of interpretations where the number of objects is at most k.

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

Proof: This follows since to falsify T a single substitution θ is sufficient and since T 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 T using the same θ. ■

Lemma 2 *Let $T \in \mathcal{H}(P)$, and let I be any interpretation. If $I \models T$ then for any set A of objects of I, $I_{|A} \models T$.*

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

2.7 The Algorithm Prop-Horn

For reference, we describe the propositional algorithm by Angluin et al. (1992) which we refer to later as Prop-Horn. We first define the basic operations of the algorithm. Let I be an interpretation, and let $\textit{prop-ant}(I)$ be the conjunction of all positive ground literals true in I, and $\textit{prop-neg}(I)$ be the set of all positive ground literals that are false in I. The set $\textit{prop-cands}(I)$ is the set of clauses $\{\textit{prop-ant}(I) \rightarrow A \mid A \in \textit{prop-neg}(I)\} \cup \{\overline{\textit{prop-ant}(I)}\}$. For a set of interpretations S, $\textit{prop-cands}(S) = \cup_{s \in S}\textit{prop-cands}(s)$.

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 the extension of predicates in the product is defined to have the intersection of the positive ground literals in I_1, I_2. We denote the intersection by $I_1 \wedge I_2$.

The algorithm is described in Table 1. The algorithm maintains an ordered set of "representative" negative examples from which it builds its hypothesis by using the *prop-cands*() operation that generates "candidate" clauses. A 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 the target expression T, and that each refinement makes progress is some measurable way (Angluin et al., 1992).

3 Some Illustrative Examples

We discuss some simple examples in order to develop an intuition for the construction that follows. The discussion here is informal and precise definitions appear in the sections that follow.

Let the signature be $P = \{p_1, \ldots, p_5\}$ where all predicates are of arity 2. First recall that our expressions are function free and hence cannot refer to constants or object names. Therefore these names are not important and we can abstract them away from examples. Consider the case where the target expression is the single clause

$$T = \forall x, y, z, (p_1(x, y) \wedge p_2(y, z) \rightarrow p_3(x, x)),$$

and the negative example (with domain $\{1, 2, 3\}$)

$$I_1 = \{p_1(1, 2), p_2(2, 3), p_4(2, 3)\}.$$

In order to show that $I_1 \not\models T$ we can substitute $1/x, 2/y, 3/z$ and satisfy the antecedent in I_1 while not satisfying the consequent. Therefore, we can find an approximation of the antecedent of the clause simply by substituting back each variable to its corresponding object in I_1. This yields $p_1(x, y) \wedge p_2(y, z) \wedge p_4(y, z)$. Notice that the literals in this antecedent are a superset of the true set of literals in T. Assuming that we have a way of finding out what the right consequent is (we will later simply try all possibilities) this process yields the clause $(p_1(x, y) \wedge p_2(y, z) \wedge p_4(y, z) \rightarrow p_3(x, x))$. Once this is done all we need to do is somehow omit the extra literal $p_4(y, z)$ to get the correct clause.

This process may encounter a problem when used with

$$I_2 = \{p_1(1, 2), p_2(2, 2), p_4(2, 3)\}.$$

In this case, the resulting clause is $(p_1(x,y) \wedge p_2(y,y) \wedge p_4(y,z) \rightarrow p_3(x,x))$ and we see that the variables y, z in the original clause have been unified into a single variable y in the resulting clause. The cause of this is the fact that the substitution $1/x, 2/y, 2/z$ showing that $I_2 \not\models T$ maps the same object to both y and z.

These two problems, extra literals and unified variables, may be solved by using several negative examples of the same clause and the direct product construction. Consider the two negative examples

$$\begin{aligned} I_3 &= \{p_1(1,2), p_2(2,2), p_4(2,3)\} \\ I_4 &= \{p_1(a,a), p_2(a,b), p_5(b,c)\}. \end{aligned}$$

Each of these interpretations will generate an extra literal and both unify variables; I_3 unifies y, z and I_4 unifies x, y. The domain of the direct product $I_3 \otimes I_4$ is the Cartesian product of those of I_3 and I_4, namely, $\{1a, 1b, 1c, 2a, 2b, 2c, 3a, 3b, 3c\}$. An atom $p(\alpha\beta, \gamma\delta)$ is true in the product if its projections $p(\alpha, \gamma)$ and $p(\beta, \delta)$ are true in the corresponding interpretations. Thus the extension of predicates in the product is

$$I_3 \otimes I_4 = \{p_1(1a, 2a), p_2(2a, 2b)\}.$$

When a clause is generated from this interpretation we get the target clause exactly. A slightly modified example shows that products may also generate new extra literals. Consider the two negative examples

$$\begin{aligned} I_5 &= \{p_1(1,2), p_1(1,3), p_2(2,2), p_4(2,3)\} \\ I_6 &= \{p_1(a,a), p_1(a,b), p_2(a,b), p_5(b,c)\}. \end{aligned}$$

The product has the same domain as above and the extension of predicates is

$$I_5 \otimes I_6 = \{p_1(1a, 2a), p_1(1a, 2b), p_1(1a, 3a), p_1(1a, 3b), p_2(2a, 2b)\}.$$

Clearly, the size of the product may be as large as the product of the sizes of the original interpretations. If using products several times the size may increase exponentially.

A final observation is that (even if the product is large) there is a small number of domain elements in the product which are of interest. These are the elements participating in a falsifying substitution for the clause. Namely, in the last product these are $1a, 2a, 2b$. We could in principle project out other domain elements and all information about them and still retain the necessary information, as in

$$(I_5 \otimes I_6)_{|\{1a,2a,2b\}} = \{p_1(1a, 2a), p_1(1a, 2b), p_2(2a, 2b)\},$$

where some of the extra literals (but not all) are removed. Note that the two remaining $p_1()$ atoms are versions of *the same* atom in I_5.

It is not easy, however, to find the right set of domain elements; an example in the next section shows that it is also not guaranteed to actually decrease the size of the interpretations (i.e. remove extra literals) which was our original aim. We also show in the next section that projection is guaranteed to be useful in case the original interpretations did not unify variables. We call such a projection *a pairing* of the interpretations. Consider the two negative examples

$$\begin{aligned} I_7 &= \{p_1(1,2), p_1(1,3), p_2(2,3), p_4(2,3)\} \\ I_8 &= \{p_1(a,b), p_1(a,c), p_2(b,c), p_5(b,c)\}. \end{aligned}$$

The product has the same domain as above and the extension of predicates is

$$I_7 \otimes I_8 = \{p_1(1a,2b), p_1(1a,2c), p_1(1a,3b), p_1(1a,3c), p_2(2b,3c)\}.$$

We can project this interpretation on $\{1a, 2b, 3c\}$ to get

$$(I_7 \otimes I_8)_{|\{1a,2b,3c\}} = \{p_1(1a,2b), p_1(1a,3c), p_2(2b,3c)\}.$$

The main difference from the previous example is that the projection set corresponds to a 1-1 matching of the domain elements of I_7, I_8. This guarantees that none of the atoms gets duplicated in the resulting projected product and its size is never larger than the original interpretations – a fact that is used in proving that our algorithm converges.

As the examples illustrate, extra literals may be removed by products or their projections. Examples which generate unified variables are harder to deal with since projections of their products may increase the size of the interpretations. Our treatment below simply rules out this situation by defining a new semantics where different variables in a clause must be bound to different domain elements in interpretations. This in turn allows us to design an algorithm for the task. Later we show that this is not too bad a restriction since we can use the resulting algorithm to solve the original problem as well.

Finally, note that we only discussed the case with a single clause. A further level of complication arises when there is more than one clause. Intuitively, the algorithm below approximates each clause in the target expression using negative examples for that clause. Naturally, the algorithm also needs to find out which clause a negative example corresponds to, and make sure that a single clause is not approximated several times by different examples. These aspects are dealt with formally in the next section.

4 Unique Substitution Semantics

Motivated by the discussion above we define an alternative semantics forcing different variables in a clause to be bound to different domain elements in interpretations. The approach we take is similar to the one by Haussler (1989) where (translated to our setting) it is shown that a single universally quantified clause is learnable from equivalence and membership queries. 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 4 *Let I be an interpretation and $C \in \mathcal{H}(P)$ be a clause with variables in X. We say that C is d-true in I (and I is a d-model of C), and denote it by $I \models_d C$ if for all 1-1 substitutions θ that map each variable to a distinct object in I, $C(\theta(X))$ is true in I, where the semantics for ground clauses remains as before.*

For $T \in \mathcal{H}(P)$ where T is a conjunction of clauses, T is d-true in I if and only if all clauses in T are d-true in I.

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 $T \models_d C_1$ then $T \models_d C_2$. Note also that Lemma 1 and Lemma 2 hold in this model as well. We modify the learning model accordingly so that equivalence and membership queries evaluate interpretations according to $\models_d$. Denote these modified oracles by $\mathrm{EQ}^{\models_d}$ and $\mathrm{MQ}^{\models_d}$.

The new semantics define the notion of d-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.

A Direct Product is an operation on interpretations that is well known for characterising Horn expressions. Products have been used before for learning and they are closely related to least general generalisations (Plotkin, 1970; Horvath & Turán, 1995; Horvath et al., 1997). 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 predicates in $\otimes(I_1, I_2, \ldots, I_j)$ 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 $I_1 \otimes I_2$. Examples for products were given in the previous section.

Products are important since they exactly characterise the class of Horn expressions. Of interest in the current context is the fact that, for propositional expressions, products become the intersection operation used by the algorithm Prop-Horn. The following theorem is essentially due to McKinsey (1943). Related results were developed by Horn (1951) and greatly expanded in model theory (Chang & Keisler, 1990).

Theorem 3 (McKinsey, 1943) *A universally quantified first order expression is equivalent (under $\models$) to a universally quantified first order Horn expression if and only if its set of models is closed under direct products.*

For $\models_d$ McKinsey's theorem does not hold. A product of two d-models of T may not be a d-model of T.[2] 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.

An operation similar to pairing has been recently discussed by Geibel and Wysotzki (1997) in the context of learning relational decision trees. The effort there is to reduce the size of the least general generalisation of clauses which is a basic operation used to construct the node tests in the tree.

Lemma 4 *Let $T \in \mathcal{H}(P)$. Then the set of d-models of T is closed under pairings.*

Proof: Let I_1, I_2 be d-models of T, and C a clause in T. Assume that a pairing I d-falsifies C and consider a substitution $\theta = (\theta_1, \theta_2)$ such that C is falsified by I with respect to θ, 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 d-falsifies the clause. ■

Corollary 5 *If J is a pairing of I_1 and I_2, and J d-falsifies $C \in \mathcal{H}(P)$ then at least one of I_1, I_2 d-falsifies C, and both d-cover C.*

The following lemma shows that for range restricted clauses, pairings characterise Horn expressions, namely the class $\mathcal{H}(P)^-$. This fact is however not needed for our result that establishes learnability of $\mathcal{H}(P)$.

Lemma 6 *Let T be a conjunction of universally quantified first order range restricted clauses. If the set of d-models of T is closed under pairings then T is equivalent (under $\models_d$) to an expression in $\mathcal{H}(P)^-$ (i.e. it is Horn).*

Proof: The proof adapts the technique of McKinsey (1943) to the current setting. Let $T = \forall X, C_1 \wedge C_2 \wedge \ldots \wedge C_s$ 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" (Selman & Kautz, 1996) 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, $T \models_d C^i$ and therefore T can be rewritten as $T = \forall X, C^i \wedge C_2 \wedge \ldots \wedge C_s$. In this way all the non-Horn clauses of T can be replaced with Horn clauses.

[2] For example let $T = (p_1(X,Y) \rightarrow p_2(X))$, $I_1 = \{p_1(1,1)\}$, and $I_2 = \{p_1(a,b), p_2(a)\}$. Both I_1 and I_2 are positive but their product $\{p_1(1a, 1b)\}$ is negative.

Table 2: The algorithm *A1*: Learn $\mathcal{H}(P)$ under $\models_d$ using EQ$^{\models_d}$ and MQ$^{\models_d}$

```
1. Maintain an ordered set of interpretations $S$, initialised to $\emptyset$ and let $H = rel\text{-}cands(S)$.
2. Repeat until $H \equiv_d T$:
   (a) Ask an equivalence query to get a counter example in case $H \not\equiv_d T$.
   (b) On a positive counter example $I$ (s.t. $I \models_d T$) remove wrong clauses (s.t. $I \not\models_d C$) from $H$.
   (c) On a negative counter example $I$ (s.t. $I \not\models_d T$):
       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 size is smaller than that of $s_i$ then
               A. (Optional Step) Minimise the number of objects in $J$ while still negative – (use MQ).
               B. Replace $s_i$ with $J$.
               C. Quit loop (Go to Step 2(c)iv)
       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 $rel\text{-}cands(s_j)$ to it.
```

To prove the claim assume that for all i, $T \not\models_d C^i$ and let I_i be a d-model of T which is not a d-model of C^i (which exists since $T \not\models_d C^i$). 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 T is range restricted, 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 with respect to $\overline{\theta}$ all negative literals of C are true in I (since the componentwise literals must be true in order to falsify 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 $I \not\models_d C$, which contradicts the fact that the d-models of T are closed under pairings. ∎

The learning algorithm *A1*, described in Table 2, is similar in structure to Prop-Horn. The algorithm generalises this scheme by using pairing (instead of intersection) and other appropriate operations. In particular, let *prop-ant*(I), *prop-neg*(I) and *prop-cands*(I) be as defined in Section 2.7, and let X be a set of variables in 1-1 correspondence to the objects of I. Then, $\mathit{rel\text{-}ant}(I), \mathit{rel\text{-}neg}(I)$ and $\mathit{rel\text{-}cands}(I)$ are derived from their propositional counterparts by substituting objects with their corresponding variables from X.

The algorithm maintains an ordered set S of negative interpretations. These are used to generate the hypothesis by using $\mathit{rel\text{-}cands}(s_i)$ for each $s_i \in S$. On a positive counter example wrong clauses (that are falsified by the example) are removed from H.

On a negative counter example the algorithm first minimises the number of objects in the counter example. This can be done greedily by removing one object at a time and asking a membership query. By Lemma 1 and Lemma 2 this yields a correct counter example that has at most k objects. The algorithm then tries to find a pairing of this counter example with one of the interpretations s_i in S that results in a negative example J which has an extension smaller than that of s_i, or a smaller number of objects. This is done by trying all possible matchings of objects in the corresponding interpretations and appealing to a membership query oracle. The first s_i for which this happens is replaced with the resulting pairing (after an optional step minimising the number of objects in J which is performed as above). In case no such pairing is found for any of the s_i, the minimised 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 the 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 by Angluin et al. (1992) 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 d-covers a clause d-falsified by it. This guarantees that two elements of S do not d-falsify the same clause of T (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 7 *Let I be a negative counter example after the minimisation of the number of objects (in Step 2(c)i). Assume that the algorithm tests s_i (in Step 2(c)ii). If there is a clause $C \in \mathcal{H}(P)$ such that $T \models_d C$, s_i d-covers C, and I d-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 d-covers C, and θ_2 a substitution showing that I d-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), d-falsifies C with respect to $\theta = (\theta_1, \theta_2)$. Therefore J is negative for T.

Since $I \not\models_d C$ 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 $\mathit{rel\text{-}cands}(s_i)$ 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 *rel-ant*(s_i). To get β, assume this mapping of variables is used, and pick the element of *rel-cands*(s_i) that has the same consequent as C. There are two cases: if the consequent of C is already in *rel-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 *rel-ant*(s_i), β is indeed in *rel-cands*(s_i). The next argument holds in both cases.

Since β can be obtained from C by adding literals to it we have that $T \models_d \beta$ 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. Note that, since C and β have the same variables, θ_2 can be used for β as well. It follows that β is not falsified by I with respect to θ_2. 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. ■

Example 1 *The following example shows that Lemma 7 does not hold under the normal semantics, thus motivating the change in semantics. The lemma shows that under the stated conditions there is a pairing of s_i and I that passes the test in Step 2(c)ii. In particular it has a smaller extension than s_i. Consider*

$$T \;=\; \forall x, y, z, (p_1(x,y)p_2(y,z) \rightarrow p_3(x)) \wedge (p_2(x,y)p_1(y,z) \rightarrow p_3(x)).$$

Let

$$I_1 = \{p_1(1,2), p_1(2,2), p_2(2,2), p_3(2)\},$$

then $s_1^1 = I_1$ (denoting versions of s by superscript), and assuming we know how to get the consequent

$$H = (p_1(x,y)p_1(y,y)p_2(y,y)p_3(y) \rightarrow p_3(x)).$$

Let

$$I_2 = \{p_1(b,b), p_2(a,b), p_2(b,b), p_3(b)\},$$

then

$$s_1^1 \otimes I_2 \;=\; \{p_1(1b,2b), p_1(2b,2b), p_2(2a,2b), p_2(2b,2b), p_3(2b)\}.$$

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

$$s_1^2 = \{p_1(1b,2b), p_1(2b,2b), p_2(2b,2b), p_3(2b)\}$$

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

$$s_1^2 = \{p_1(2b, 2b), p_2(2a, 2b), p_2(2b, 2b), p_3(2b)\}$$

which is a dual case. In either case the size is the same as that of s_1^1. In addition if the algorithm replaces s_i with a pairing of the same size then it may be tricked into an infinite loop: by using $I_3 = I_1$ we get that $s_1^2 \otimes I_3$ is isomorphic to $s_1^1 \otimes I_2$.

Lemma 8 *At all times in the algorithm, for all k, i such that $k < i$, and for all $C \in \mathcal{H}(P)$ such that $T \models_d C$, if s_i d-falsifies C then s_k does not d-cover C.*

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 d-falsifies C, and by Lemma 7 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 Lemma 2 if the optional Step 2(c)iiA is taken, then if J d-falsifies C then it did so also before the minimisation. By Corollary 5, since J d-falsifies C, s_k d-covers C, and this contradicts the inductive assumption.

Consider next the case where s_i is replaced by J. Again, if J d-falsifies C then it did so also before the optional step. Therefore (again by Corollary 5) both s_i and the counter example I, d-cover C, and at least one d-falsifies C. If s_i d-falsifies C we get a contradiction to the inductive assumption. If I d-falsifies C then by Lemma 7 s_k is replaced if tested. ∎

Theorem 9 *The class $\mathcal{H}(P)$ is learnable (under $\models_d$) by the algorithm A1 using $EQ^{\models_d}$ and $MQ^{\models_d}$. For $T \in \mathcal{H}^k(P)$ with m clauses, the algorithm makes at most $E_N + E_P$ equivalence queries and $(2n + mk^k)E_N$ 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 d-falsifies at least one clause of T. By Lemma 8, no two elements d-falsify the same clause of T 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 some s_i, or introduces a new element s_i. The size 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 $2n$ 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 $\mathit{rel\text{-}cands}(s_i)$ 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 T (and it is removed), then clearly $A \rightarrow C$ is not implied by T. It can be seen that for a fixed i, the antecedents of clauses in $\mathit{rel\text{-}cands}(s_i)$ 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 (Angluin et al., 1992).

5 Extensions

In this section we apply Theorem 9 to other settings. In doing so we omit the exact bounds which can be easily derived. A related discussion and comparison of various models of learning when queries are not allowed is given by De Raedt (1997).

5.1 Normal Semantics

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

Lemma 10 *For every $T \in \mathcal{H}^k(P)$ with m clauses there is an expression $U(T) \in \mathcal{H}^k(P)$ with at most mk^k clauses such that for all interpretations I, $I \models T$ if and only if $I \models_d U(T)$.*

Proof: We construct $U(T)$ from T by considering every clause separately. For a clause C in T with j variables generate a 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. The required $U(T)$ is the conjunction of all clauses generated for all clauses of T. The construction makes sure that all possible ways to falsify a clause C in T 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. ■

Table 3: The algorithm *A2*: Learn $\mathcal{H}(P)$ under $\models$ using EQ and MQ

- Run algorithm *A1* using EQ and MQ instead of $\text{EQ}^{\models_d}$ and $\text{MQ}^{\models_d}$.
- The hypothesis H of *A1* can be interpreted according to $\models$.
- Modify the treatment of positive counter examples in *A1*. On counter example I, remove a clause C from H if $I \not\models C$.

Hence the algorithm working under $\models_d$ can simply interact with oracles working according to $\models$ and still learn the same class. Moreover, the algorithm can be modified so that it uses hypotheses can be interpreted according to $\models$ rather than $\models_d$. For this the hypothesis itself need not be changed syntactically. Instead the same expression is used with respect to $\models$. If this is done then negative counter examples ($I \not\models T$ and $I \models H$) remain counter examples since $I \not\models_d U(T)$ (by Lemma 10) and $I \models_d H$ (since $I \models C$ implies $I \models_d C$). For positive counter examples ($I \models T$ and $I \not\models H$), we have that $I \models_d U(T)$ (by Lemma 10) but it may be the case that $I \models_d H$ and I is not a counter example. To handle this the algorithm needs to be modified to evaluate clauses according to $\models$ when removing clauses on a positive counter example. We call this modified algorithm *A2*; the modifications are summarised in Table 3.

Naturally, it must be verified the the theorem still holds. For this note that the only place where the hypothesis is used is in the proof of Lemma 7, where we must argue that β is not removed from the hypothesis. In this case we know that the antecedent of β is a superset of the antecedent of a clause in $U(T)$ and their consequent is identical. It follows that $T \models \beta$. Therefore, $I \models T$ implies $I \models \beta$ and β is not removed by the modified process.

Corollary 11 *The class $\mathcal{H}(P)$ is learnable (under $\models$) by the algorithm A2 using EQ and MQ.*

5.2 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 $\mathcal{H}^k(P)$ under $\models_d$ can simulate $\mathcal{H}^k(P, =)$ under $\models$ hence yielding a learning result for $\mathcal{H}(P, =)$ under $\models$.

Lemma 12 *For every $T \in \mathcal{H}^k(P, =)$ with m clauses there is an expression $U^*(T) \in \mathcal{H}^k(P)$ with at most mk^k clauses such that for all interpretations I, $I \models T$ if and only if $I \models_d U^*(T)$.*

Proof: We first show that for each $T \in \mathcal{H}(P, =)$ there is an expression G equivalent to T under $\models$ such that no clause in G includes inequalities and G has the same number of clauses as T. (Hence inequalities are in some sense useless.) To get G from T 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 $I \not\models T$ 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 and 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 T 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 $U^*(T)$ 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 10 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 $U^*(T)$.

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 C^* 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 C^* is falsified by I under the substitution induced from θ (which is 1-1 for its variables), implying $I \not\models_d C^*$.

Finally, assume $I \not\models_d U^*(T)$ for some I. Thus some clause C' in $U^*(T)$ 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 $I \not\models G$. ■

Here again, the hypothesis of the algorithm can be converted into an expression in the class being learned. This time there is no need to modify the algorithm, but the hypothesis is syntactically modified. In particular, this can be done by adding equalities

Table 4: The algorithm *A3*: Learn $\mathcal{H}(P,=)$ under $\models$ using EQ and MQ

- Run algorithm *A1* using EQ and MQ instead of $\text{EQ}^{\models_d}$ and $\text{MQ}^{\models_d}$.
- The hypothesis H of *A1* is modified syntactically by adding equalities on all variables used in the clauses. It can be interpreted according to $\models$.

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 $(p(x,y)p(y,z)(x \neq y)(x \neq z)(y \neq z) \rightarrow q(z))$. This is summarised as algorithm *A3* in Table 4

Corollary 13 *The class $\mathcal{H}(P,=)$ is learnable (under $\models$) by the algorithm A3 using EQ and MQ.*

5.3 Entailment Queries

For learning from entailment (Frazier & Pitt, 1993) examples are clauses in the language. Here we use ground clauses as examples. This seems natural and corresponds to what is done in inductive logic programming. It is easy to see that the same ideas apply if examples are universally quantified clauses.

In the following, we modify the signature of the language so as to include an infinite number of constant symbols (as the natural numbers) in direct correspondence with the possible names of domain elements in the interpretations; thus the signature is (P, N), and in all interpretations the domain is a subset of N.[3] The expressions in $\mathcal{H}(P)$ still do not include constant symbols. We need however to accommodate the constant symbols in the interpretations for (P, N). Let I be an interpretation for the signature P and assume that the domain is a subset of the natural numbers. Then I is converted into I^N, an interpretation for (P, N), by letting the domain be N and mapping each constant i to the object i. The extension of predicates is the same as in I. Clearly, no atom $p(\ldots, i, \ldots)$ such that i is not in the domain of I is true in I^N. Similarly, from an interpretation I^N for (P, N) we get an interpretation I for P by omitting all objects that do not appear in the extension of any predicate in I^N. Note that the operations above are not complementary since an interpretation I for P may include objects that do not appear in the extension of any predicate.

Lemma 14 *Let T be a conjunction of range restricted clauses over P.*

1. *For every interpretation I for P, $I \models T$ if and only if $I^N \models T$.*
2. *For every interpretation I^N for (P, N), $I^N \models T$ if and only if $I \models T$.*

[3] Clearly, any other generic set of names can be used as long as constants and object names are equated.

Proof: For range restricted clauses all variables in a clause appear in the antecedent. Therefore objects that do not appear in the extension of any atom in an interpretation cannot participate in a falsifying substitution (since otherwise the antecedent is falsified). Such objects can therefore be added or removed without changing the truth value of T in the interpretation. ■

Since the semantics does not change we use I and assume that the corresponding conversion is done whenever needed. We can now define the learning model. For *Entailment Membership Queries* (EntMQ), the learner presents $C(\theta(X))$, a ground instance of a clause $C \in \mathcal{H}$ (i.e. all variable are substituted to constants) and the oracle returns "yes" iff $T \models C$. For *Entailment Equivalence Queries* (EntEQ) the learner presents a hypothesis $H \in \mathcal{H}$ and the oracle returns "yes" if $H = T$ and otherwise it returns a ground clause c that is a counter example ($T \models c$ and $H \not\models c$ or vise versa).

We first observe that membership queries can be replaced with entailment queries. Recall that *prop-cands*(I) is the propositional operation of Section 2.7.

Lemma 15 *Let I be an interpretation and $T \in \mathcal{H}(P)^-$. Then $I \not\models T$ if and only if for some $c \in$ prop-cands(I), $T \models c$.*

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

For the other direction assume $I \not\models T$. Therefore it falsifies some clause C of T 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 *prop-ant*(I)), and its consequent is not. Now consider the clause c whose antecedent is *prop-ant*(I) and whose consequent is identical to the consequent of c'. Then c is in *prop-cands*(I) and $T \models C \models c' \models c$. ■

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 T. Moreover, if entailment queries are available we can make sure when creating a hypothesis that its clauses are always implied by T. This can be done by asking an entailment query for each of the clauses in *rel-cands*(s_i). Since clauses in *rel-cands*$()$ are universally quantified they must be translated to ground clauses if we want to use EntMQ. This can be done by substituting an arbitrary distinct constant for every variable in the clause. Note that by performing this we avoid the n^k dependence in the running time (needed for positive counter examples). These modifications are summarised as Algorithm *A4* in Table 5.

Corollary 16 *The class $\mathcal{H}(P)^-$ is learnable (under $\models$) by the algorithm A4 using EQ and EntMQ.*

For EntEQ notice that if $T \models H$ and c is a counter example clause for H then it is the case that $T \models c$ and $H \not\models c$. Namely, only one type of counter examples is encountered.

Table 5: The algorithm *A4*: Learn $\mathcal{H}(P)^-$ under $\models$ using EQ and EntMQ

- Run algorithm *A2* simulating its oracles as needed and modified as follows.
- Modify the hypothesis generation step of *A2*. For each clause that *A2* intends to include in the hypothesis, if $C \notin \mathcal{H}(P)^-$ then do not include C. Otherwise, include C only if $T \models C$.

  To test this, substitute an arbitrary distinct constant for every variable in C to get $C(\theta)$. Use EntMQ to test whether $T \models C(\theta)$. Include C in H if the answer is "yes".
- Calls to the oracle EQ are treated as in *A2*.
- On a call to MQ with interpretation I answer "no" if and only if for some $c \in$ *prop-cands*(I), $T \models c$ (use EntMQ to test this).

Lemma 17 *Let $T, H \in \mathcal{H}_D(P)^-$. If $T \models H$ and c is a counter example clause for H then an interpretation I such that $I \not\models T$ and $I \models H$ can be found in time $O(|H||P|n^k n^a)$ where n is the number of objects in c.*

Proof: Let I be an interpretation that includes the objects in c, and where the extension of predicates includes precisely the positive literals in the antecedent of c. The idea (Frazier & Pitt, 1993; Reddy & Tadepalli, 1998) is to compute the "closure" of I with respect to H. Clearly, $I \not\models T$. 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 all clauses in H are definite and H is satisfied by the interpretation that has all ground atoms true. We claim that I satisfies the conditions of the lemma.

Since $H \not\models c$, the consequent of c is never added to I (since forward chaining is sound) and hence $I \not\models T$. 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. ■

The above process is similar to the use of the chase procedure to decide on uniform containment of database queries (Sagiv, 1988). The lemma implies that EntEQ can be used to simulate EQ if we restrict the hypothesis to include only clauses in $\mathcal{H}_D(P)^-$. The modifications are summarised as Algorithm *A5* in Table 6.

Corollary 18 *The class $\mathcal{H}_D(P)^-$ is learnable (under $\models$) by the algorithm A5 using EntEQ and EntMQ.*

5.4 Inductive Logic Programming

Both interpretations and clauses were used as examples in ILP and the results above are therefore relevant to this area. The main other feature incorporated into ILP models is

Table 6: The algorithm *A5*: Learn $\mathcal{H}_D(P)^-$ under $\models$ using EntEQ and EntMQ

- Run algorithm *A4* simulating its oracles as needed and modified as follows.
- Modify the hypothesis generation step of *A4*. For each clause that *A4* intends to include in the hypothesis, include C if and only if $C \in \mathcal{H}_D(P)^-$.
- Calls to the oracle EntMQ are treated as in *A4*.
- On a call to EQ with modified hypothesis H, first call EntEQ with H. On a counter example c (such that $T \models c$ and $H \not\models c$) run the procedure of Lemma 17 to generate an interpretation I and present I as a counter example to *A4*.

the use of background knowledge. The background knowledge B is an expression over the signature (P, N) which is known to the learner before the learning session. A distinction has been made between *extensional background knowledge* where B is a conjunction of positive ground literals, and *intentional background knowledge* where B may include general Horn expressions. We show that both types can be handled by our algorithm in the case of learning from entailment.

We consider the setting as defined by Cohen (1995b). In this setting, an example is meant as a positive example for some concept in the world (which is the consequent in some clause in T). 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 T with respect to B if and only if $T \wedge B \wedge D \models E$. Since this is equivalent to $T \wedge B \models (D \rightarrow E)$ we see that a positive ILP example is similar to an example clause.

We modify the learning model accordingly. The target $T \in \mathcal{H}_D(P)^-$ and the background knowledge B are fixed by an adversary. The target is hidden from the learner but B is given to the learner. For a *ILP equivalence oracle* (ILPEQ), the learner presents a hypothesis H, and the oracle returns "yes" if and only if $T \wedge B$ is equivalent to $H \wedge B$. Otherwise it returns a ILP counter example, namely, a pair (E, D) which is positive for one of T or H but not the other. A *ILP membership oracle* (ILPMQ) for B and T when presented with a pair (E, D) answers "yes" if and only if $T \wedge B \models (D \rightarrow E)$.

For the extensional case, the idea is to run the same algorithm but modify the examples slightly. Since $T \wedge B \models (D \rightarrow E)$ if and only if $T \models (B \wedge D \rightarrow E)$, a ILP counter example (E, D) can be turned into a clause $c = (B \wedge D \rightarrow E)$ which is a counter example for H. The use of EntEQ is therefore straightforward.

For EntMQ a clause c can be seen as a pair corresponding to $c = (D \rightarrow E)$; the only problem one needs to get around in order to apply Corollary 18 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. Namely, we want to have $T \models (D \rightarrow E)$ if and only if $T \wedge B \models (D \rightarrow E)$. This can be done by using distinct new object names in the queries (that do not appear

Table 7: The algorithm *A6*: Learn $\mathcal{H}_D(P)^-$ under $\models$ relative to an extensional background knowledge using ILPEQ and ILPMQ.

- Run algorithm *A5* simulating its oracles as needed.
- On a call to EntEQ with hypothesis H, first call ILPEQ with H. On a counter example (E, D) (such that $T \wedge B \models (D \to E)$ and $H \wedge B \not\models (D \to E)$), present $c = (B \wedge D \to E)$ to *A5* as a counter example.
- On a call to EntMQ with clause c, rename the constants in c so that none of them appears in B to get $c' = (D \to E)$. Present (E, D) to ILPMQ and return the same answer to *A5*.

in B). The resulting algorithm *A6* is summarised in Table 7.

Corollary 19 *The class $\mathcal{H}_D(P)^-$ is learnable (under $\models$) relative to an extensional background knowledge by the algorithm A6 using ILPEQ and ILPMQ.*

For intentional background knowledge in $\mathcal{H}_D(P)^-$ the idea is to have the algorithm learn the expression $T \wedge B \in \mathcal{H}_D(P)^-$. The background knowledge can be incorporated into the algorithm by using the clauses in B to initialise the set S of interpretations the algorithm uses. For each clause C, we generate an interpretation s_C from its antecedent by substituting a unique object to each variable. Let the set of interpretations so generated be S_0. Clearly, the clause C is in *rel-cands*(s_C). However, since the order of elements in S is important, we must take care when using these interpretations. The solution we present uses one of the previous learning algorithms as a subroutine that automatically adapts to this requirement.

The learning algorithm *A7* runs in two phases. In the first phase it uses *A4* as follows. On an EntMQ for the clause $c = (D \to E)$, it presents (E, D) to ILPMQ and answers accordingly. On a EQ with hypothesis H, it evaluates H on all the interpretations in S_0. If for some $s \in S_0$, $s \models H$, then s is returned to *A4* as a counter example. Otherwise the algorithm moves to the second phase.

In the second phase it runs *A5* but using the set S resulting from the first phase to initialise the set S of *A5*. EntMQ are dealt with as in the first phase. On an EntEQ with hypothesis H, the algorithm presents H to ILPEQ to get a counter example (E, D) that it returns as a counter example $c = (D \to E)$ to *A5*. The algorithm is summarised in Table 8.

It is easy to see that after the first phase and throughout the second one the hypothesis of the algorithm will satisfy $H \models B$ and $T \wedge B \models H$. Therefore the complexity of learning in the second stage is bounded by the complexity of learning T alone in the case with no background knowledge. The number of EntMQ in the first phase can be bounded using the number of counter examples in this phase (which is at most the number of clauses in B) and the analysis as above.

Table 8: The algorithm *A7*: Learn $\mathcal{H}_D(P)^-$ under $\models$ relative to an intentional background knowledge using ILPEQ and ILPMQ.

1. Compute S_0 from the clauses of B.
2. Run algorithm *A4* simulating its oracles as needed and restricting the hypothesis to $\mathcal{H}_D(P)^-$.
   - On a call to EntMQ with clause $c = (D \rightarrow E)$ present (E, D) to ILPMQ and return the same answer to *A4*.
   - On a call to EntEQ with hypothesis H, evaluate H on interpretations in S_0; if an interpretation $I \in S_0$ such that $I \models H$ is found then return it as a counter example to *A4*. Otherwise go to Step3.
3. Run algorithm *A5* initialising S to be the same as in the last stage of *A4*, and simulating its oracles as needed.
   - On a call to EntMQ with clause $c = (D \rightarrow E)$ present (E, D) to ILPMQ and return the same answer to *A5*.
   - On a call to EntEQ with hypothesis H, first call ILPEQ with H. On a counter example (E, D) (such that $T \wedge B \models (D \rightarrow E)$ and $H \wedge B \not\models (D \rightarrow E)$), present $c = (D \rightarrow E)$ to *A5* as a counter example.

Corollary 20 *The class $\mathcal{H}_D(P)^-$ is learnable (under $\models$) relative to an intentional background knowledge by the algorithm A7 using ILPEQ and ILPMQ.*

Clearly, both kinds of background knowledge can be combined though this does not allow for clauses that include both variables and constants.

5.5 Learning to Reason

We next show that the algorithm for learning from entailment is robust in the sense that if the target expression is not Horn then it will find a Horn expression which is as close to it as possible. In fact we show that the algorithm is a Learn to Reason algorithm (Khardon & Roth, 1997) with respect to the class $\mathcal{H}_D(P)^-$. This is formalised using the notion of least upper bounds that were introduced by Selman and Kautz (1996) and discussed by various authors (e.g. Frazier & Pitt, 1993; Khardon & Roth, 1996; Del Val, 1996).

Definition 5 *Let $\mathcal{G}, \mathcal{H}$ be classes of first order expressions over the signature P. An expression $T \in \mathcal{H}$ is the least upper bound of $G \in \mathcal{G}$ in $\mathcal{H}$, if (1) $G \models T$, and (2) for all $T' \in \mathcal{H}$ such that $G \models T'$, it is the case that $G \models T \models T'$.*

In the following we fix $\mathcal{G}$ to be the class of expressions composed of conjunctions of range restricted clauses, and $\mathcal{H}$ to be $\mathcal{H}_D(P)^-$. By Lemma 14, this allows us to use the signature (P, N) without modifying the semantics. Since $\mathcal{H}_D(P)^-$ is closed under conjunctions it is easy to see that the least upper bound is well defined and unique. We modify the learning model so that the target expression is in $\mathcal{G}$ but examples are (ground clauses) in $\mathcal{H}$. For EntMQ, the learner presents $C(\theta(X))$, a ground instance of a clause $C \in \mathcal{H}_D(P)^-$ (i.e. all variable are substituted to constants) and the oracle returns "yes" iff $T \models C(\theta(X))$. For EntEQ, the learner presents a hypothesis $H \in \mathcal{H}(P)^-$ and the oracle returns "no" if there is a clause $C \in \mathcal{H}(P)^-$ and a substitution θ such that $T \models C(\theta(X))$ and $H \not\models C(\theta(X))$ or vise versa. In this case it returns such a ground clause as a counter example. Otherwise it returns "yes". Notice that it may be the case that $G \neq H$ but there is no counter example in $\mathcal{H}(P)^-$ and the oracle returns "yes". We denote this restricted oracle by EntEQ$[\mathcal{H}_D(P)^-]$.

Theorem 21 *Given access to EntMQ and EntEQ$[\mathcal{H}_D(P)^-]$ and for any target expression $G \in \mathcal{G}$, algorithm A5 will find an expression $H \in \mathcal{H}_D(P)^-$ that is equivalent (under $\models$) to the least upper bound of G in $\mathcal{H}_D(P)^-$.*

Proof: Let T be the least upper bound. The theorem follows by observing that the oracles behave as if the algorithm was learning T. In particular, by the definition of least upper bounds, for any $C \in \mathcal{H}_D(P)^-$, $G \models C$ if and only if $T \models C$, and hence EntMQ are answered correctly according to T. For EntEQ, recall that the algorithm makes sure that $G \models H$ where H is the hypothesis. Therefore, counter examples are such that $G \models c$ and $H \not\models c$. Now since T is the least upper bound $G \models T \models c$, and c is a counter example for T as well. ■

This result can be translated into an "on-line" learning scenario where the learner uses its hypothesis to reason about the world (as expressed by G). When it makes a mistake it finds a counter example clause that it can use to refine its hypothesis. Our result implies that even if G is not Horn, and despite the fact that we do not have an algorithm to learn $\mathcal{G}$, the learning algorithm can learn a representation that supports correct reasoning with respect to G for all expressions in $\mathcal{H}_D(P)^-$. Results of this type were previously developed for propositional logic and called Learning to Reason; Theorem 21 generalises the Learning to Reason result for propositional Horn expressions (Theorem 7.1 in Khardon & Roth, 1997). Finally, we note that since our analysis in Section 4 discussed clauses C such that $T \models C$ rather than clauses in the expression T, a similar claim can be made for learning from interpretations under a suitable restriction of EQ.

6 A Lower Bound

In this section we characterise the Vapnik-Chevonenkis dimension (VC-Dim) of $\mathcal{H}(P)$. It is known that the VC-Dim of a concept class is a lower bound for the number of equivalence and membership queries when learning this class (Maass & Turán, 1992). The following theorem thus shows that $\Omega(m|P|k^a)$ queries are necessary; the only scope for improvement is therefore in the factor k^k.

We start with the necessary definitions (Blumer et al., 1989; Maass & Turán, 1992). Let A be a set, $\mathcal{B} \subseteq 2^A$, and $S \subseteq A$. Then $\Pi_{\mathcal{B}}(S) = \{B \cap S \mid B \in \mathcal{B}\}$ is the set of subsets of S that can be obtained by intersection with elements of $\mathcal{B}$. If $|\Pi_{\mathcal{B}}(S)| = 2^{|S|}$ then we say that $\mathcal{B}$ shatters S. Finally, VC-Dim($\mathcal{B}$) is the size of the largest set shattered by $\mathcal{B}$ (or ∞ if arbitrary large sets are shattered).

In our case A is the set of interpretations, and $\mathcal{B}$ is the class $\mathcal{H}(P)$ interpreted under $\models$. Let $\mathcal{H}^k(P)[m]$ be the class of expressions in $\mathcal{H}^k(P)$ with at most m clauses.

Theorem 22 *If $|P| \geq 2(k+1+\log m)$ and all predicates in P have arity a then VC-Dim($\mathcal{H}^k(P)[m]$) $= \Theta(m|P|k^a)$.*

Proof: Let $\alpha = |P|k^a$, and fix any k variables; the number of positive literals generated by predicates in P with these variables is at most α. The number of antecedents is thus bounded by 2^α and the number of consequents by α. Therefore,

$$|\mathcal{H}^k(P)[m]| \leq \sum_{i=1}^{m} \binom{\alpha 2^\alpha}{i} \leq (\frac{e\alpha 2^\alpha}{m})^m = O(2^{\alpha m})$$

and VC-Dim($\mathcal{H}^k(P)[m]$) $\leq \log(|\mathcal{H}^k(P)[m]|) = O(\alpha m)$.

For the lower bound we assume for simplicity that there are $k+1+\log m$ unary predicates, $L_0, L_1, \ldots, L_k, N_1 \ldots, N_{\log m}$, and the rest of the predicates are of arity a.

We first show that $\mathcal{H}^k(P)[1]$ can shatter a set S of size $\Omega(\alpha)$. The domain in all interpretations in S is $\{1, \ldots, k\}$, and in all interpretations the extension of $L_1, \ldots, L_k$ is precisely $L_1(1), \ldots, L_k(k)$ and $L_0, N_1, \ldots, N_{\log m}$ have an empty extension.

Let Q be the set of ground atoms that can be generated by the non-unary predicates in P over the domain $\{1, \ldots, k\}$. Each interpretation in S will omit exactly one element of Q. Note that $|S| = |Q|$ and that if $|P| \geq 2(k+1+\log m)$ then $|Q| \geq \frac{1}{2}|P|k^a$.

To see that this set of interpretations is shattered by $\mathcal{H}^k(P)[1]$ note that using the conjunction $L_1(x_1) \wedge \ldots \wedge L_k(x_k)$ in the antecedent of a clause we can make sure that in any falsifying substitution x_i is bound to i for all i.

Let S be a subset of the interpretations to be rejected (falsified) by a single clause. The required clause is of the form $C \to L_0(x_1)$, where C is the conjunction of all atoms that are true in all elements of S where element i is substituted with the variables x_i. By construction this includes the conjunction $L_1(x_1) \wedge \ldots \wedge L_k(x_k)$. Now, for each $s \in S$ the antecedent is falsified by s using the obvious substitution, and therefore the clause is falsified by s. For $s \notin S$ the clause is not falsified since the atom of Q missing in s appears in the antecedent of the clause (with the corresponding variables substituted for the objects).

For $\mathcal{H}^k(P)[m]$, we replace each of the interpretations above with m interpretations. This is done by using the $N()$ predicates to give a label between 0 and $m-1$ to each generated interpretation. In particular, for each $0 \leq i \leq m-1$ generate an interpretation by adding exactly one of the atoms $N_j(1), N_j(2)$ for each j according to the binary encoding of i. Now, given a set S to be rejected, first divide it into m subsets according to the label. Each subset can be rejected using the clause as above where we add the encoding of the label to the antecedent. ■

7 Using the Propositional Algorithm

In this section we show that the algorithm Prop-Horn can be applied more directly to the relational learning problem. The resulting algorithm is similar to *A2*. The result here is slightly less good than the one using *A2* both in terms of the class learnable which is $\mathcal{H}(P)^-$ and the mistake bound (only slightly worse). It may be of interest however since the proof is different and is based essentially on a reduction to the propositional case.

We first show that if the domain is fixed then $\mathcal{H}(P)^-$ 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 T be a universally quantified Horn expression on a set of variables $X = (X_1, \ldots, X_k)$

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

Let $\theta_1, \ldots, \theta_{k^k}$ be an enumeration of all possible mappings of k variables to objects in an interpretation with domain $\{1, \ldots, k\}$. Consider the propositional expression

$$\begin{aligned} T_p \;=\; & C_1(\theta_1(X))C_1(\theta_2(X))\ldots C_1(\theta_{k^k}(X)) \\ & C_2(\theta_1(X))C_2(\theta_2(X))\ldots C_2(\theta_{k^k}(X)) \\ & \ldots \\ & C_m(\theta_1(X))C_m(\theta_2(X))\ldots C_m(\theta_{k^k}(X)), \end{aligned}$$

where we have omitted the conjunction symbols. 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. As in Lemma 14 we have:

Lemma 23 *Let $T \in \mathcal{H}^k(P)^-$, $I \in \mathcal{I}^k$, and let T_p be the propositional version of T described above. Then the following conditions are equivalent:*
(1) $I \not\models T$
(2) $inflate(I) \not\models T$
(3) $inflate(I) \not\models T_p$.

Proof: Clearly (1) implies (2) and (3) since the falsifying substitution in I suffices. Now (3) implies (2) since the clause falsified in T_p supplies the falsifying substitution for T. To see that (2) implies (1) notice that phantom assignments do not change the truth value of range restricted clauses. 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. ■

Lemma 23 suggests that $\mathcal{H}(P)^-$ can be learned by using the propositional algorithm directly. 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 1 and Lemma 2) or using $inflate()$ to set the number of objects to k (relying on Lemma 23). This however does not quite work if arbitrary examples rather than examples in $\mathcal{I}^k$ are used since T_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.

Assume first that $T \in \mathcal{H}^k(P)^-$ and the algorithm knows the correct value of k. The algorithm *A8* runs Prop-Horn using $\mathcal{I}^k$ as the domain and simulating its oracles while interacting with the first order oracles. The algorithm uses Prop-Horn's set of interpretations S to generate its own hypothesis, $H = \wedge_{s_i \in S} \textit{rel-cands}(s_i)$. (In fact, only only range restricted clauses in *rel-cands*() need to be included in the hypothesis.) Initially $S = \emptyset$ and H is true on any interpretation.

When Prop-Horn asks a membership query (after computing the intersection of x with an element $s_i \in S$) 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 $H = \wedge_{s_i \in S} \textit{rel-cands}(s_i)$ 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.

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 (as in *A2*) greedily by removing one object at a time and asking a membership query. By Lemma 1 and Lemma 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 *A8*. Finally, 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. The algorithm is summarised in Table 9.

Theorem 24 *The class $\mathcal{H}(P)^-$ is learnable (under $\models$) by the algorithm A8 using EQ and MQ. For $T \in \mathcal{H}^k(P)^-$ 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.*

Table 9: The algorithm *A8*: Learn $\mathcal{H}(P)^-$ using EQ and MQ (propositional version).

1. Let $k = 1$.
2. Run algorithm Prop-Horn with domain $\{1, \ldots, k\}$ simulating its oracles as needed.
3. On a call to EQ (propositional with fixed k) with the set S generating Prop-Horn's hypothesis, compute $\textit{rel-cands}(S)$ as a hypothesis. Present H to EQ to get a counter example I.
    - if I is a positive counter example ($I \models T$) then remove wrong clauses (s.t. $I \not\models C$) from H.
    - If I is a negative counter example, then minimise the number of objects in I using MQ (as in *A2*). If I has at most k objects then return $inflate(I)$ as a counter example to Prop-Horn.

      Otherwise, let $k = \max\{2k, \text{number of objects in } I\}$ and restart Step 2.
4. On a call to MQ (propositional with fixed k) with interpretation I, present I to MQ and answer in the same way.

We first show that if entailment queries are also allowed then the algorithm can be used to learn the class $\mathcal{H}(P)^-$. For this we modify algorithm *A8* as follows. The set $\textit{rel-clauses}(I)$ is the set of clauses $\{C \in \textit{rel-cands}(I) \mid T \models C\}$. Given a set S of interpretations, $\textit{rel-clauses}(S)$ can be computed by appealing to an entailment oracle (as before by substituting constants to variables). Notice that since $s_i \not\models T$, s_i falsifies at least one of the clauses of T and hence $\textit{rel-clauses}(s_i)$ is not empty. The hypothesis of the algorithm is now computed by $H = \wedge_{s_i \in S} \textit{rel-clauses}(s_i)$.

Lemma 25 *The class $\mathcal{H}(P)^-$ is learnable (under $\models$) by the modified A8 algorithm using EQ, MQ, and EntMQ. For $T \in \mathcal{H}^k(P)^-$ 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 $T \models H$, and therefore a counter example is such that $I \not\models T$ and $I \models H$.

Lemma 23 identifies a target expression T_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 T_p, (2) if Prop-Horn asks an equivalence query and if $H \neq T$ 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 expression T_p. Part (1) follows immediately by Lemma 23.

For (2) note that if $H \neq T$ 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 23 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. Here we only consider clauses c such that $T_p \models c$, and therefore have $T_p \models h$. 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.

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 time complexity of the algorithm is similar to that of *A2*. The number of queries is governed by the query complexity of Prop-Horn which is polynomial in the number of propositional variables and the size of T_p. The latter is $O(mk^k)$ where $T \in \mathcal{H}^k(P)^-$ has m clauses.

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 be the maximum of $2k$ and the number of objects in I, where I is the counter example, and restart the algorithm. Correctness follows since as long as we do not meet counter examples that are too large, the propositional learning problem simulates the learning of T when restricted to interpretations of size k. (Essentially the construction of T_p can be generalised to have i^k substitutions when considering i objects.) We therefore have at most $\log k$ iterations where in each iteration the complexity is bounded as before. ■

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

Proof: [Proof of Theorem 24] The modified *A8* 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 $H = \wedge_{s_i \in S} \textit{rel-cands}(s_i)$, the algorithm may get positive counter examples, that are used to remove 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 $\textit{rel-cands}(s_i)$ the same bounds follow (essentially the entailment queries are traded for positive counter examples). ■

8 Concluding Remarks

We have shown that universally quantified function free Horn expressions are learnable in several models of exact learning from queries. This includes learning from interpretations, learning from entailment, learning with intensional or extensional background knowledge and learning to reason. 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.

The algorithms presented are polynomial in the number of predicate symbols in the language and the number of clauses in the target Horn expression but exponential in the arity of predicates and the number of universally quantified variables. We also derived lower bounds for these tasks by way of characterising the VC-dimension of this class of expressions. This shows that apart for the dependency on on the number of variables our algorithms are essentially optimal.

In order to develop the results we introduced the unique substitutions semantics and the pairing operation that enabled us to control the size of generalised clauses. The pairing operation as well as the operations of omitting one object at a time from interpretation while using MQ can be seen as refined forms of minimising the size of interpretations or the relevant clauses. A "fine grain" minimisation by omission of one atom at a time is used in the propositional domain for example by Angluin (1988) for learning monotone DNF, and for relational problems by Reddy and Tadepalli (1997, 1998). Work by Aizenstein and Pitt (1995) indicates that this may not always be successful. Our work identifies more "coarse grain" minimisation steps that are safe for function free expressions.

The application of these ideas in a practical ILP system would require an interactive setting where membership queries are answered. Some work in ILP included systems with similar requirements (Sammut & Banerji, 1986; Muggleton & Buntine, 1992) and our results can be applied in these scenarios. Clearly, finding heuristics for reducing the number of queries is an important step in this direction.

There are several natural questions as for improvements of these results. These include for example allowing constants and function symbols in the learned expressions, improving the complexity or proving better lower bounds, and allowing for alternation of quantifiers. Another aspect concerns the learning model. Shapiro's (1983) system introduced the model inference problem, where a learner is trying to find a logic program corresponding to an "intended interpretation". There are subtle differences between this requirement and the ones studied in this paper. In addition the set of queries available to the learner is also different. Clarifying these aspects will be of interest. Finally, some connections of the problems studied here to work in database theory have been mentioned and further exploration of these may prove useful.

When considering function symbols in the language it is important to make sure that learned expressions are useful in the sense that computations with them are decidable and efficient. Some progress in this direction was recently made, showing that a natural generalisation of range restricted expressions, where every term that appears in the consequent of a clause also appears in its antecedent, is learnable (Khardon, 1998).

Acknowledgements

A preliminary version of this paper appeared in COLT 1998. This work was partly supported by EPSRC Grant GR/M21409. 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.

I am grateful to several colleagues for their contribution: Dan Roth for discussions regarding the notion of products and Mark Jerrum, Chandra Reddy, and Prasad Tadepalli for comments on earlier drafts.

References

Aizenstein, H., & Pitt, L. (1995). On the learnability of disjunctive normal form formulas. *Machine Learning*, *19*, 183–208.

Angluin, D. (1988). Queries and concept learning. *Machine Learning*, *2*(4), 319–342.

Angluin, D., Frazier, M., & Pitt, L. (1992). Learning conjunctions of Horn clauses. *Machine Learning*, *9*, 147–164.

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, *36*(4), 929–865.

Chang, C., & Keisler, J. (1990). *Model Theory.* Amsterdam Oxford : North-Holland.

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

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

De Raedt, L. (1997). Logical settings for concept learning. *Artificial Intelligence*, *95*(1), 187–201. See also relevant Errata.

De Raedt, L., & Bruynooghe (1992). An overview of the interactive concept learner and theory revisor CLINT. In Muggleton, S. (Ed.), *Inductive Logic Programming.* Academic Press.

De Raedt, L., & Dzeroski, S. (1994). First order *jk*-clausal theories are PAC-learnable. *Artificial Intelligence*, *70*, 375–392.

Del Val, A. (1996). Approximate knowledge compilation: the first order case. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 498– 503 Portland, Oregon. AAAI Press.

Dzeroski, S., Muggleton, S., & Russell, S. (1992). PAC-learnability of determinate logic programs. In *Proceedings of the Conference on Computational Learning Theory*, pp. 128–135 Pittsburgh, PA. ACM Press.

Frazier, M., & Pitt, L. (1993). Learning from entailment: An application to propositional Horn sentences. In *Proceedings of the International Conference on Machine Learning*, pp. 120–127 Amherst, MA. Morgan Kaufmann.

Geibel, P., & Wysotzki, F. (1997). A logical framework for graph theoretical decision tree learning. In *International Workshop on Inductive Logic Programming*, pp. 173–180 Prague, Czech Republic. Springer. LNAI 1297.

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

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

Horvath, T., Sloan, B., & Turán, G. (1997). Learning logic programs by using the product homomorphism method. In *Proceedings of the Conference on Computational Learning Theory*, pp. 10–20 Nashville, Tennessee. ACM Press.

Horvath, T., & Turán, G. (1995). Learning logic programs with structured background knowledge. In *The Fifth Workshop on Inductive Logic Programming*.

Khardon, R. (1996). Learning to take actions. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 787–792 Portland, Oregon. AAAI Press.

Khardon, R. (1998). Learning range-restricted Horn expressions. Tech. rep. ECS-LFCS-98-395, Laboratory for Foundations of Computer Science, Edinburgh University.

Khardon, R., & Roth, D. (1996). Reasoning with models. *Artificial Intelligence*, *87*(1-2), 187–213.

Khardon, R., & Roth, D. (1997). Learning to reason. *Journal of the ACM*, *44*(5), 697–725.

Lloyd, J. (1987). *Foundations of Logic Programming.* Springer Verlag. Second Edition.

Maass, W., & Turán, G. (1992). Lower bound methods and separation results for on-line learning models. *Machine Learning*, *9*, 107–145.

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

Minker, J. (Ed.). (1988). *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann.

Muggleton, S., & Buntine, W. (1992). Machine invention of first order predicates by inverting resolution. In Muggleton, S. (Ed.), *Inductive Logic Programming*. Academic Press.

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

Muggleton, S., & Feng, C. (1992). Efficient induction of logic programs. In Muggleton, S. (Ed.), *Inductive Logic Programming*. Academic Press.

Papadimitriou, C. H., & Yannakakis, M. (1997). On the complexity of database queries. In *Proceedings of the symposium on Principles of Database Systems*, pp. 12–19 Tucson, Arizona. ACM Press.

Plotkin, G. D. (1970). A note on inductive generalization. In Meltzer, B., & Michie, D. (Eds.), *Machine Intelligence 5*, pp. 153–163. American Elsevier.

Reddy, C., & Tadepalli, P. (1997). Learning Horn definitions with equivalence and membership queries. In *International Workshop on Inductive Logic Programming*, pp. 243–255 Prague, Czech Republic. Springer. LNAI 1297.

Reddy, C., & Tadepalli, P. (1998). Learning first order acyclic Horn programs from entailment. In *International Conference on Inductive Logic Programming*, pp. 23–37 Madison, WI. Springer. LNAI 1446.

Reddy, C., Tadepalli, P., & Roncagliolo, S. (1996). Theory guided empirical speedup learning of goal decomposition rules. In *International Conference on Machine Learning*, pp. 409–416 Bari, Italy. Morgan Kaufmann.

Sagiv, Y. (1988). Optimizing datalog programs. In Minker, J. (Ed.), *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann.

Sammut, C., & Banerji, R. (1986). Learning concepts by asking questions. In Michalski, R., Carbonell, J., & Mitchell, T. (Eds.), *Machine Learning : An AI Approach, Volume II*. Morgan Kaufman.

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

Shapiro, E. Y. (1983). *Algorithmic Program Debugging*. MIT Press, Cambridge, MA.

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

Valiant, L. G. (1985). Learning disjunctions of conjunctions. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pp. 560–566 Los Angeles, CA. Morgan Kaufmann.