Learning Range Restricted Horn Expressions*

Roni Khardon

Division of Informatics
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ
Scotland
roni@dcs.ed.ac.uk

Abstract. We study the learnability of first order Horn expressions from equivalence and membership queries. We show that the class of range restricted Horn expressions, where every term in the consequent of every clause appears also in the antecedent of the clause, is learnable. The result holds both for the model where interpretations are examples (learning from interpretations) and the model where clauses are examples (learning from entailment).
The paper utilises a previous result on learning function free Horn expressions. This is done by using techniques for flattening and unflattening of examples and clauses, and a procedure for model finding for range restricted expressions. This procedure can also be used to solve the implication problem for this class.

1 Introduction

We study the problem of exactly identifying universally quantified first order Horn expressions using Angluin's [Ang88] model of exact learning. Much of the work in learning theory has dealt with learning of Boolean expressions in propositional logic. Early treatments of relational expressions were given by [Val85,Hau89], but only recently more attention was given to the subject in framework of Inductive Logic Programming [MDR94,Coh95a,Coh95b]. It is clear that the relational learning problem is harder than the propositional one and indeed except for very restricted cases it is computationally hard [Coh95b]. To tackle this issue in the propositional domain various queries and oracles that allow for efficient learning have been studied [Val84,Ang88]. In particular, propositional Horn expressions are known to be learnable in polynomial time from equivalence and membership queries [AFP92,FP93]. In the relational domain, queries have been used in several systems [Sha83,SB86,DRB92,MB92] and results on learnability in the limit were derived [Sha91,DRB92]. More recently progress has been made on the problem of learning first order Horn expressions from equivalence and membership queries using additional constraints or other additional queries [Ari97,RT97,Kha98,RT98,RS98].

* This work was partly supported by EPSRC Grant GR/M21409.

In particular [Kha98] shows that universally quantified function free Horn expressions are exactly learnable in several models of learning from equivalence and membership queries. This paper extends these results to a class of expressions allowing the use of function symbols. In particular, we present algorithms for learning *range restricted* Horn expressions where every term in the consequent of a clause appears also (possibly as a subterm) in the antecedent of the clause. In fact, our results hold for a more expressive class, *weakly range restricted* Horn expressions, that allows for some use of equalities in the antecedent of a clause.

Several kinds of examples have been considered in the context of learning first order expressions. 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 [DR97] and were both studied before. We present algorithms for learning weakly range restricted Horn expressions in both settings. We also show that the implication problem for such expressions is decidable, and provide an upper bound for its complexity. This motivates the use of this class since learned expressions can be used as a knowledge base in a system in a useful way.

The result for learning from interpretations is derived by exhibiting a reduction to the function free case, essentially using flattening - replacing function symbols with predicate symbols of arity larger by one [Rou92]. The reduction uses flattening and unflattening of examples and clauses, and an axiomatisation of the functionality of the new predicates. Learning from entailment is then shown possible by reducing it to learning from interpretations under the given restrictions. This relies on a procedure for model finding for this class, which also proves the decidability of inference for it. We also derive learnability results for range restricted expressions as corollaries. Interestingly, despite the use of reduction, for learning from entailment we can use range restricted expressions as the hypothesis language, but for learning from interpretations hypotheses are weakly range restricted.

The rest of the paper is organised as follows. The next section provides preliminary definitions. Section 3 presents range restricted expressions and some of their basic properties. Sections 4 and 5 develop the results on learning from interpretations and learning from entailment respectively, and Section 6 discusses the implication problem. The concluding section discusses related work and directions for future work.

2 Preliminaries

2.1 First Order Horn Expressions

We follow standard definitions of first order expressions; for these see [CK90,Llo87]. The learning problems under consideration assume a pre-fixed known and finite

signature $\mathcal{S}$ of the language. That is, $\mathcal{S} = (P, F)$ where P is a finite set of predicates, and F is a finite set of function symbols, each with its associated fixed arity. Constants are simply 0-ary function symbols and are treated as such. In addition a set of variables $x_1, x_2, x_3, \ldots$ is used to construct expressions.

We next define *terms* and their depth. A variable is a term of depth 0. A constant is a term of depth 0. If $t_1, \ldots, t_n$ are terms, each of depth at most i (and one with depth precisely i) and $f \in F$ is a function symbol of arity n, then $f(t_1, \ldots, t_n)$ is a term of depth $i + 1$.

An *atom* is an expression $p(t_1, \ldots, t_n)$ where $p \in P$ is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms. An atom is called a *positive literal*; a *negative literal* is an expression $\neg l$ where l is a positive literal. A clause is a disjunction of literals where all variables are taken to be universally quantified. A Horn clause has at most one positive literal and an arbitrary number of negative literals. A Horn clause $\neg p_1 \vee \ldots \vee \neg p_n \vee p_{n+1}$ is equivalent to its "implicational form" $p_1 \wedge \ldots \wedge p_n \rightarrow p_{n+1}$. When presenting a clause in this way we call $p_1 \wedge \ldots \wedge p_n$ the *antecedent* of the clause and p_{n+1} the *consequent* of the clause. A Horn expression is a conjunction of Horn clauses.

The truth value of first order expressions is defined relative to an interpretation I of the predicates and function symbols in $\mathcal{S}$ [Llo87]. Interpretations are also called *structures* in model theory [CK90] and we use these terms interchangeably. An interpretation I includes a domain D which is a (finite) set of elements. For each function symbol $f \in F$ of arity n, I associates a mapping from D^n to D; if $f(a_1, \ldots, a_n)$ is associated with a we say that $f(a_1, \ldots, a_n)$ *corresponds to* a *in* I. For each predicate symbol $p \in P$ of arity n, I specifies the truth value of p on n-tuples over D. The *extension* of a predicate in I is the set of positive instantiations of it that are true in I. In structural domains [Hau89,Kha96,RTR96], domain elements are objects in the world and an instantiation describes properties and relations of objects. We therefore refer to domain elements as *objects*. Let $str(\mathcal{S})$ be the set of structures (interpretations) for the signature $\mathcal{S}$.

The truth value of an expression in an interpretation I is defined in a standard way [CK90,Llo87]. Note that a Horn clause is not true (falsified) in I iff there is a variable assignment (a substitution) that simultaneously satisfies the antecedent and falsifies the consequent. The terms (1) T is true in I, (3) I satisfies T, (4) I is a model of T, and (5) $I \models T$, have the same meaning. Let $T_1, T_2 \in \mathcal{H}(S, =)$ 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.

2.2 The Learning Model

We define here the scheme of learning from interpretations [DRD94]. Learning from entailment [FP93], where examples are clauses in the language is defined in Section 5. An example is an interpretation; an example I is positive for a target expression T if $I \models T$ and negative otherwise. Examples of this form have been used before by various authors including [Hau89,DRD94,RT97,Kha98].

We use Angluin's model of learning from Equivalence Queries (EQ) and Membership Queries (MQ) [Ang88]. Let $\mathcal{H}$ be a class under consideration, $\mathcal{H}'$

a (possibly different) class used to represent hypotheses, and let $T \in \mathcal{H}$ be the target expression. For membership queries, the learner presents an interpretation I and the oracle MQ returns "yes" iff $I \models T$. For equivalence queries, the learner presents a hypothesis $H \in \mathcal{H}'$ and the oracle EQ returns "yes" if for all I, $I \models T$ iff $I \models H$; otherwise it returns a counter example I such that $I \models T$ and $I \not\models H$ (a positive counter example) or $I \not\models T$ and $I \models H$ (a negative counter example).

In the learning model, $T \in \mathcal{H}$ is fixed by an adversary and hidden from the learner. The learner has access to EQ and MQ and must find an expression H equivalent to T (under the definition above). If there is an algorithm that performs this task we say that $\mathcal{H}$ *is learnable with hypothesis in* $\mathcal{H}'$, or, when $\mathcal{H}' = H$, just $\mathcal{H}$ *is learnable*. For complexity we measure the running time of the algorithm and the number of times it makes queries to EQ and MQ. It is known [Lit88,Ang88] that learnability in this model implies pac-learnability [Val84].

3 Range Restricted Horn Expressions

Definition 1. (definite clauses) *A clause is definite if it includes precisely one positive literal. For a signature $\mathcal{S}$, let $\mathcal{H}(S)$ be the set of Horn expressions over $\mathcal{S}$ in which all clauses are definite.*

Definition 2. (range restricted clauses) *A definite Horn clause is called range restricted*[1] *if every term that appears in its consequent also appears in its antecedent, possibly as a subterm of another term. For a signature $\mathcal{S}$, let $\mathcal{H}_R(\mathcal{S})$ be the set of Horn expressions over $\mathcal{S}$ in which all clauses are definite and range restricted.*

For example, the clause $(p_1(f_1(f_2(x)), f_3()) \rightarrow p_2(f_2(x), x))$ is range restricted, but the clause $(p_1(f_1(f_2(x)), f_3()) \rightarrow p_2(f_1(x), x))$ is not. We also consider clauses with a limited use of equality in their antecedent.

Definition 3. (equational form) *A definite clause C with equalities in its antecedent and where every non-equational literal includes only variables as terms, and every equational literal is of the form $(x_{i_{n+1}} = f(x_{i_1}, \ldots, x_{i_n}))$ where $f \in F$ and x_{i_j} are variables is in equational form. For a signature $\mathcal{S}$, let $\mathcal{H}(S, =)$ be the set of Horn expressions over $\mathcal{S}$ in which all clauses are definite and in equational form.*

Every range restricted clause can be transformed into an equational form by unfolding terms bottom-up and replacing them with variables. Formally, for $T \in \mathcal{H}_R(\mathcal{S})$, transform each clause C in T as follows. Find a term $f(x_1, \ldots, x_n)$ in C all of whose sub-terms are variables (this includes constants) and rewrite

[1] A similar restriction has been used before by several authors. Unfortunately, in a previous version of [Kha98] it was called "non-generative" while in other work it was called "generative" [MF92]. The term "range-restricted" was used in database literature for the function free case [Min88]. Here we use a natural generalisation for the case with function symbols.

the clause by replacing all occurrences of this term with a new variable z, and adding a new literal $(z = f(x_1, \ldots, x_n))$ to the antecedent of C. For example the clause

$$p_1(x_1, f_1(x_2)) \wedge p_2(f_2()) \rightarrow p_1(x_1, f_2())$$

is first transformed (using $f_1(x_2)$) into

$$(z_1 = f_1(x_2)) \wedge p_1(x_1, z_1) \wedge p_2(f_2()) \rightarrow p_1(x_1, f_2())$$

and then (using the constant $f_2()$) into

$$((z_1 = f_1(x_2)) \wedge (z_2 = f_2()) \wedge p_1(x_1, z_1) \wedge p_2(z_2) \rightarrow p_1(x_1, z_2)).$$

In the equational form of range restricted clauses each new variable z_i has one defining equation, and we may think of the variables involved in the equation as its ancestors. For example, x_2 is an ancestor of z_1 in the example above. Constructed in this way all variables that appear in equational literals are ancestors of some variable in the original literals. Since the clause is range restricted this holds for variables in the consequent as well. We will consider the case where z_i may have more than one such equation as in

$$((z_1 = f_3(x_1)) \wedge (z_1 = f_1(x_2)) \wedge (z_2 = f_2()) \wedge p_1(x_1, z_1) \wedge p_2(z_2) \rightarrow p_1(x_1, z_2))$$

but where the variables in equations are still ancestors in this sense.

Definition 4. (root variables, legal ancestor) *Let C be a definite Horn clause in equational form. (1) The variables appearing in non-equational literals in the antecedent are called* root variables. *(2) Root variables are* legal ancestors. *(3) If an equational literal $(z = f(x_1, \ldots, x_n))$ appears in the antecedent and z is a legal ancestor then $x_1, \ldots, x_n$ are also legal ancestors.*

Definition 5. (weakly range restricted clauses) *A definite Horn clause in equational form is called weakly range restricted if every variable that appears in its consequent or in equational literals is a legal ancestor. For a signature $\mathcal{S}$, let $\mathcal{H}_R(\mathcal{S}, =)$ be the set of Horn expressions over $\mathcal{S}$ in which all clauses are definite and weakly range restricted.*

The following proposition (proof omitted) shows that we can replace range restricted expressions with their equational form.

Proposition 1. *Let $T \in \mathcal{H}_R(\mathcal{S})$ and let T' be the equational form of T computed as above then for all $I \in str(\mathcal{S})$, $I \models T$ if and only if $I \models T'$.*

We next define legal objects in interpretations to play a role similar to legal ancestors in clauses.

Definition 6. (legal objects) *Let $I \in str(\mathcal{S})$, and let D be the domain of I. (1) If $p(a_1, \ldots, a_n)$ is true in I for $p \in P$ then $a_1, \ldots, a_n$ are legal objects. (2) If $f(a_1, \ldots, a_n)$ is mapped to a_{n+1} in I where $f \in F$ and a_{n+1} is a legal object then $a_1, \ldots, a_n$ are legal objects.*

The main property of weakly range restricted expressions used in our constructions is the fact that their truth value of is not effected by non-legal objects.

Lemma 1. *Let $I \in str(\mathcal{S})$, let C be a weakly range restricted clause, and let θ be a mapping of the variables of C into objects of I. If $I \not\models C\theta$ then all objects mapped by θ are legal objects.*

Proof. Assume that θ maps a variable of C to a non-legal object. If the equational literals in the antecedent of C are not satisfied in I by θ, then $I \models C\theta$.

Notice that if a variable x is mapped to a non-legal object in I and $(z = f(\ldots, x, \ldots))$ is true in I then z is also mapped to a non-legal object. Now, since every variable is an ancestor of some root variable in C, if the equational literals of C are satisfied in I by θ, then some root variable is mapped to a non-legal object by θ. By definition this implies that an atom $p(\ldots)$ in the antecedent of C is made false by θ. Therefore $I \models C\theta$. □

4 Learning from Interpretations

We first define a modified signature of the language above. Similar transformations have been previously used under the name of flattening [Rou92] (see also [NCDW97]). For each function symbol f of arity n, define a new predicate symbol f_p of arity $n+1$. Let F_p be the new set of predicates so defined, $\mathcal{S}' = (P \cup F_p, \emptyset)$ be the modified signature, $\mathcal{H}_R(\mathcal{S}')$ the set of range restricted (and function free) Horn expressions over the predicates in $P \cup F_p$, and $str(\mathcal{S}')$ be the set of interpretations for $\mathcal{S}'$.

Reductions appropriate for learning with membership queries were defined in [AK95] where they are called pwm-reductions. Three transformations are required. The *representation transformation* maps $T \in \mathcal{H}_R(\mathcal{S})$ to $T' \in \mathcal{H}_R(\mathcal{S}')$, the *example transformation* maps $I \in str(\mathcal{S})$ to $I' \in str(\mathcal{S}')$, and the *membership queries transformation* maps $I' \in str(\mathcal{S}')$ to $\{Yes, No\} \cup str(\mathcal{S})$. Intuitively, the learner for $T \in \mathcal{H}_R(\mathcal{S})$ will be constructed out of a learner for $T' \in \mathcal{H}_R(\mathcal{S}')$ (the image of the representation transformation) by using the transformations. The obvious properties required of these transformations guarantee correctness. The example and representation transformations guarantee that the learner receives correct examples for T' and the membership query transformation guarantees that queries can be either answered immediately or transferred to the membership oracle for T.

The Representation Transformation: Let $T \in \mathcal{H}(S, =)$ be a Horn expression, then the expression $flat(T) \in \mathcal{H}_R(\mathcal{S}')$ is formed by replacing each equational literal $(z = f(x_1, \ldots, x_n))$ with the corresponding atom $f_p(x_1, \ldots, x_n, z)$. Thus, the equational clause given above is transformed to:

$$f_{3_p}(x_1, z_1) \wedge f_{1_p}(x_2, z_1) \wedge f_{2_p}(z_2) \wedge p_1(x_1, z_1) \wedge p_2(z_2) \rightarrow p_1(x_1, z_2).$$

The definitions of root variables and legal ancestors hold for the flat versions as well. We also axiomatise the fact that the new predicates are functional. Our

treatment diverges from previous uses of flattening [Rou92] in that the function symbols are taken out of the language. For every $f \in F$ of arity n let

$$\begin{aligned} exist_f &= (\forall x_1, \forall x_2 \ldots, \forall x_n, \exists z, f_p(x_1, \ldots, x_n, z)) \\ unique_f &= (\forall x_1, \forall x_2 \ldots, \forall x_n, \forall z_1, \forall z_2, \\ &\qquad f_p(x_1, \ldots, x_n, z_1) \wedge f_p(x_1, \ldots, x_n, z_2) \rightarrow (z_1 = z_2)) \end{aligned}$$

Let $\phi_f = exist_f \wedge unique_f$, $\phi_F = \wedge_{f \in F} \phi_f$, and $\mathcal{A}_{unique} = \wedge_{f \in F} unique_f$. We call $exist_f$ the existence clause of f and $unique_f$ the uniqueness clause.

The Example Transformation: Let I be an interpretation for $\mathcal{S}$, then *flat*(I) is an interpretation for $\mathcal{S}'$ defined as follows. The domain of *flat*(I) is equal to the domain of I and the extension of predicates in P is the same as in I. The extension of a predicate $f_p \in F_p$ of arity $n+1$ is defined in a natural way to include all tuples $(a_1, \ldots, a_n, a_{n+1})$ where a_i are domain elements and $f(a_1, \ldots, a_n)$ corresponds to a_{n+1} in I.

Lemma 2. *For all $T \in \mathcal{H}(S, =)$ and for all $I \in str(\mathcal{S})$:*
(1) flat$(I) \models \phi_F$.
(2) $I \models T$ if and only if flat$(I) \models$ flat(T).

Proof. Since each constant and each term are mapped to precisely one domain element in I, part (1) is true by the construction of *flat*(I). For (2) note that *flat*(T) and the equational form of T have the same variables, and I and *flat*(I) have the same domain. Let θ be a mapping of these variables to the domain. By construction, $(z = f(x_1, \ldots, x_n))\theta$ is true in I if and only if $f_p(x_1, \ldots, x_n, z)\theta$ is true in *flat*(I). Moreover, predicates in P have the same extension in I and *flat*(I). Therefore, a falsifying substitution for one can be used as a falsifying substitution for the other. □

The Membership Queries Transformation: A mapping converting structures from $str(\mathcal{S}')$ to $str(\mathcal{S})$ is a bit more involved. Let $J \in str(\mathcal{S}')$; if $J \models \phi_F$ then the previous mapping can simply be reversed, and we denote it by *unflat*(J). Otherwise there are two cases. If J falsifies the uniqueness clause, it is in some sense inconsistent with the intension for usage of the functional predicates. Such interpretations are not output by the algorithm of [Kha98] when learning $\mathcal{H}_R(\mathcal{S}')$ and hence we do not need to deal with them. If J satisfies the uniqueness clause (of all function symbols) but falsifies the existence axiom then some information on the interpretation of the function symbols is missing. In this case we complete it by introducing a new domain element $*$ and defining *complete*$(J) \in str(\mathcal{S}')$ to be the interpretation in which every ground instance of the existence clauses which is false in J is made true by adding a positive atom whose last term is $*$. For example, if there is no b such that $f_p(1, b)$ is true in J then we add $f_p(1, *)$ to *complete*(J). For any $J \in str(\mathcal{S}')$ such that $J \models \mathcal{A}_{unique}$, the interpretation J is transformed in this way into *unflat*(*complete*(J)). Note that the object $*$ is non-legal (cf. Definition 6) in *unflat*(*complete*(J)).

Lemma 3. *For all $T \in \mathcal{H}_R(\mathcal{S},=)$ and for all $J \in str(\mathcal{S}')$ such that $J \models \mathcal{A}_{unique}$ the following conditions are equivalent.*
(1) $J \models flat(T)$.
(2) $complete(J) \models flat(T)$.
(3) $unflat(complete(J)) \models T$.

Proof. Since $unflat(complete(J))$ is in $str(\mathcal{S})$ and $flat(unflat(complete(J))) = complete(J)$, Lemma 2 implies that (2) and (3) are equivalent. Now, if $J \not\models flat(T)\theta$ for some θ then since the $complete()$ construction does not change the truth value for atoms whose objects are in J we also have $complete(J) \not\models flat(T)\theta$. Thus (2) implies (1).

Finally, if $unflat(complete(J)) \not\models T\theta$ then by Lemma 1, θ does not use non-legal objects, and in particular the object $*$. Hence we can use θ in J and the argument in Lemma 2 shows that $J \not\models flat(T)\theta$. Therefore (1) implies (3). □

For $\mathcal{S} = (P, F)$ let $|\mathcal{S}| = |P| + |F|$ be the number of symbols in the signature.

Theorem 1. *The class $\mathcal{H}_R(\mathcal{S},=)$ is learnable from equivalence and membership queries with hypothesis in $\mathcal{H}_R(\mathcal{S}')$.*

For $T \in \mathcal{H}_R(\mathcal{S},=)$ with m clauses and at most t variables, the algorithm makes $O(mt^{t+2r}|\mathcal{S}|^2)$ equivalence queries, $O((nm+m^2t^t)|\mathcal{S}|t^r)$ membership queries, and its running time is polynomial in $n^t+t^t+m+|\mathcal{S}|+t^r$, where n is the number of objects in the largest counter example presented to the algorithm, and r is the maximal arity of predicates and function symbols in $\mathcal{S}$.

Proof. The theorem follows from properties of pwm-reductions [AK95] and the result in [Kha98] showing that $\mathcal{H}_R(\mathcal{S}')$ is learnable. The idea is that when learning $T \in \mathcal{H}_R(\mathcal{S},=)$ we will run the algorithm *A2* from [Kha98] to learn the expression $flat(T) \in \mathcal{H}_R(\mathcal{S}')$. When *A2* presents $H \in \mathcal{H}_R(\mathcal{S}')$ to an equivalence query we interpret this by saying that $I \in str(\mathcal{S})$ is a model of H if and only if $flat(I) \models H$. Hence given a counter example I we simply compute $flat(I)$ and present it as a counter example to *A2*. Lemma 2 and the above interpretation guarantee that the examples it receives are correct. When *A2* presents J for a membership query, we compute $unflat(complete(J))$, present it to MQ and return its answer to *A2*. Lemma 3 guarantees that the answer is correct. By Corollary 11 of [Kha98] we get that *A2* will find an expression equivalent to $flat(T)$. The complexity bound follows from [Kha98]. □

4.1 Modifying the Hypothesis Language

The previous theorem produces a hypothesis in $\mathcal{H}_R(\mathcal{S}')$ while the target expression is in $\mathcal{H}_R(\mathcal{S},=)$. We next show how to use a hypothesis in $\mathcal{H}_R(\mathcal{S},=)$.

We first need to describe the hypothesis of the learning algorithm *A2* from [Kha98]. The algorithm maintains a set of interpretations $S \subseteq str(\mathcal{S}')$ such that for each $J \in S$, $J \not\models flat(T)$. The hypothesis is $H = \wedge_{J \in S} rel\text{-}cands(J)$ where $rel\text{-}cands(J)$ is a set of clauses produced as follows. First take the conjunction of positive atoms true in J as an antecedent and an atom false in J as a consequent.

Each such choice of consequent generates a ground clause. Considering each ground clause separately, substitute a unique variable to each object in the clause to get a clause in *rel-cands*(J).

We generate clauses over $\mathcal{S}$ by reversing the *flat*() operation; namely, replacing every literal $f_p(x_1, \ldots, x_n, x_{n+1})$ (where $f_p \in F_p$) by the corresponding literal $(x_{n+1} = f(x_1, \ldots, x_n))$. For $C \in$ *rel-cands*(J) let *unflat*(C) be the resulting clause. Notice that *unflat*(C) is in $\mathcal{H}(S, =)$ but it may not be in $\mathcal{H}_R(\mathcal{S}, =)$ since some of the variables in its equality literals may not be legal ancestors (cf. Definition 4). Since the clauses in question are in $\mathcal{H}(S, =)$, and since *flat*(*unflat*$(C)) = C$, the following is a special case of Lemma 2.

Lemma 4. *For all $I \in str(\mathcal{S})$, for all $J \in str(\mathcal{S}')$, and for all $C \in$ rel-cands(J), $I \models$ unflat(C) if and only if flat$(I) \models C$.*

When applied in this way we can see that a hypothesis modified by *unflat*() attracts precisely the same counter examples and we get learnability with expressions over the signature $\mathcal{S}$. A further improvement is needed to generate a hypothesis in $\mathcal{H}_R(\mathcal{S}, =)$. Define legal objects of interpretation over $\mathcal{S}'$ in accordance with the definition for $\mathcal{S}$ (so that the same thing results when flattening).

Let $J \in str(\mathcal{S}')$ be an interpretation with domain D. For $D' \subset D$ let $J_{|D'}$ be the projection of J over D'. Namely, the interpretation where the domain is D' and an atom $q(a_1, \ldots, a_n)$, where $a_1, \ldots, a_n \in D'$, is true in $J_{|D'}$ if and only if it is true in J.

Lemma 5. *Let $T \in \mathcal{H}_R(\mathcal{S}, =)$ and $J \in str(\mathcal{S}')$, such that $J \models \mathcal{A}_{unique}$. Let D be the domain of J and let $a \in D$ be a non-legal object in J. Then unflat(complete$(J)) \models T$ if and only if unflat(complete$(J_{|\{D \setminus a\}})) \models T$.*

Proof. Since a is non-legal in J if and only if it is non-legal in *unflat*(*complete*(J)), Lemma 1 implies that if *unflat*(*complete*$(J)) \not\models T\theta$ then θ does not use a. Similarly, θ does not use the object $*$. Since the extension of predicates and mapping of functions over the other objects is not changed it follows that *unflat*(*complete*$(J_{|\{D \setminus a\}})) \not\models T\theta$.

For the other direction, if *unflat*(*complete*$(J_{|\{D \setminus a\}})) \not\models T\theta$ then by the same argument θ does not use the object $*$. Again, since the extension of predicates and mapping of functions over the other objects is not changed, θ can be used in *unflat*(*complete*$(J))$ without changing the truth value of T. □

Since a membership query of the algorithm (i.e. whether $J \models$ *flat*$(T))$ is translated to a membership query for T (i.e. whether *unflat*(*complete*$(J)) \models T$) the lemma indicates that *all* non-legal objects can be dropped from J before making the membership query. This fact is utilised in the next section.

For our current purpose it suffices to observe that in *A2* dropping of objects happens by default. In particular, whenever the algorithm *A2* (with its optional step taken) puts an interpretation J into the set S (that generates its hypothesis as discussed above), it makes sure that $J \not\models$ *flat*(T) and for every object a in the domain D of J, it holds that $J_{|\{D \setminus a\}} \models$ *flat*(T). If this does not hold then it

uses $J_{|\{D\setminus a\}}$ instead of J. Therefore, by Lemma 5 we get that all objects in all interpretations in S are legal objects. This in turn implies that the hypothesis is in $\mathcal{H}_R(\mathcal{S},=)$.

Theorem 2. *The class $\mathcal{H}_R(\mathcal{S},=)$ is learnable from equivalence and membership queries.*

For a clause $C \in \mathcal{H}_R(\mathcal{S})$, by *the number of distinct terms in C* we mean the number of distinct elements in the set of all terms in C and all their subterms. For example, $(p(x, f_1(x), f_2(f_1(x)), f_3()) \rightarrow q(f_1(x)))$ has 4 distinct terms $x, f_3(), f_1(x), f_2(f_1(x))$.

Corollary 1. *The class $\mathcal{H}_R(\mathcal{S})$ is learnable from equivalence and membership queries with hypothesis in $\mathcal{H}_R(\mathcal{S},=)$. The complexity is as in the previous theorem where t is the maximal number of distinct terms in a clause in the target expression.*

Proof. Learnability follows since by Proposition 1 every $T \in \mathcal{H}_R(\mathcal{S})$ has an equivalent expression in $\mathcal{H}_R(\mathcal{S},=)$. It remains to observe that each distinct term in a clause $C \in \mathcal{H}_R(\mathcal{S})$ is mapped to a variable in the equational form. □

5 Learning from Entailment

In this model examples are clauses in the underlying language $\mathcal{H}$ [FP93]. An example $C \in \mathcal{H}$ is positive for $T \in \mathcal{H}$ if $T \models C$. The equivalence and membership oracles are defined accordingly. For membership queries, the learner presents a clause C and the oracle EntMQ returns "yes" iff $T \models C$. For equivalence queries, the learner presents a hypothesis $H \in \mathcal{H}'$ and the oracle EntEQ returns "yes" if for all I, $I \models T$ iff $I \models H$; otherwise it returns a counter example C such that $T \models C$ and $H \not\models C$ (a positive counter example) or $T \not\models C$ and $H \models C$ (a negative counter example).

Since one can identify non-legal objects and (by Lemma 5) drop them before making a membership query, the following lemma indicates that we can replace MQ by EntMQ for clauses in $\mathcal{H}_R(\mathcal{S},=)$.

Lemma 6. *Let $T \in \mathcal{H}_R(\mathcal{S},=)$ and let $J \in str(\mathcal{S}')$ be such that $J \models \mathcal{A}_{unique}$ and all objects in J are legal objects. Then $unflat(complete(J)) \not\models T$ if and only if $T \models unflat(C)$ for some $C \in rel\text{-}cands(J)$.*

Proof. Let $I = unflat(complete(J))$. First note that by construction $I \not\models unflat(C)$ for all $C \in rel\text{-}cands(J)$. Hence if $T \models unflat(C)$ for some such C then $I \not\models T$.

For the other direction, let γ be the (reverse) substitution that is used when generating $rel\text{-}cands(J)$. Let R be a clause in T and θ a substitution such that $I \not\models R\theta$. The antecedent of R is satisfied by θ in I and, by Lemma 1, θ does not use the object $*$. Therefore $ant(R)\theta\gamma \subseteq ant(unflat(C))$ for all $C \in rel\text{-}cands(J)$, where $ant()$ refers to the antecedent part of the clause considered as a set of literals. (The resulting substitution $\theta\gamma$ is a variable renaming that may unify

several variables into one.) Since in $\mathit{rel\text{-}cands}(R)$ all range restricted consequents are considered, we get that for some $C' \in \mathit{rel\text{-}cands}(J)$, $R\theta\gamma \subseteq \mathit{unflat}(C')$, where here we consider a clause as a set of literals. (In other words R, θ-subsumes $\mathit{unflat}(C')$ [Plo70].) We therefore get that $T \models R \models R\theta\gamma \models \mathit{unflat}(C')$. □

The following lemma provides a model finding algorithm for $\mathcal{H}_R(\mathcal{S}, =)$.

Lemma 7. *Given $H \in \mathcal{H}_R(\mathcal{S}, =)$ and a clause $C \in \mathcal{H}_R(\mathcal{S}, =)$ such that $H \not\models C$, one can find an interpretation $I \in \mathit{str}(\mathcal{S})$ such that $I \models H$ and $I \not\models C$ in time $O(|H| \cdot |\mathcal{S}| \cdot n^{t+r})$ where $|H|$ is the number of clauses in H, n is the number of terms in C, t is the maximal number of variables in a clause of H, and r is the maximal arity.*

Proof. The idea is to generate an interpretation from C and then make sure (by forward chaining) that it is a model of H but not of C.

Generate a structure $I_0 \in \mathit{str}(\mathcal{S})$ as follows. First, introduce a unique domain element for each term in C and then join elements if their terms are equated in the antecedent of C; let this domain be D.[2] The extension of predicates in I_0 includes precisely the atoms corresponding to the antecedent of C and the mapping of domain elements produced. Let p be the (ground atom which is the) consequent of C under this mapping. The mapping of function symbols includes the initial mapping used when constructing D. It is then extended (as in the *complete*() construction) by adding another domain element $*$ and mapping each term $f(a_1, \ldots, a_n)$ that has not yet been assigned to $*$. Note that $*$ is a non-legal object.

Next, let $I = I_0$ and run forward chaining on H adding positive atoms to I. That is, repeat the following procedure: find a clause C in H and a substitution θ such that $I \not\models C\theta$ and add the atom corresponding to the consequent of $C\theta$ to I. This results in an interpretation I whose domain size is at most the number of distinct terms in C plus 1, and which is a model of H. This is true since H is definite and the domain of I_0 is finite and hence by adding atoms to I_0 we eventually get to a state where all clauses are satisfied (there is a finite number of atoms that can be added). We claim that p is not in I and hence $I \not\models C$.

Since $H \in \mathcal{H}_R(\mathcal{S}, =)$, by Lemma 1 if $I \not\models H\theta$ then θ does not use the object $*$. Hence forward chaining does not produce any positive atoms containing the object $*$. Inductively, this shows that no such atom is true in I.

Let J be some interpretation such that $J \models H$ and $J \not\models C$ (which exists by the condition of the lemma). Let θ be such that $J \not\models C\theta$ and let q be the consequent of $C\theta$. Clearly, q is not true in J. Moreover, there is a mapping from objects in I_0 (apart from $*$) to the objects in J that are used in $C\theta$, so that all positive atoms true in I_0 are true in J under this mapping, and all equalities true in I_0 (apart from ones referring to $*$) are true in J under this mapping. Namely, a homomorphic embedding [CK90] of $\mathit{flat}(I_0)_{|D}$ into $\mathit{flat}(J)$.

[2] Note that there is no need to perform equational reasoning here and a syntactic matching suffices. This is true since in $\mathcal{H}_R(\mathcal{S}, =)$ all terms are of depth 0 or 1.

Finally, assume that p is in I. Since its forward chaining does not use the object $*$, we can use the same chaining under the homomorphism to generate q in J, and therefore since $J \models H$, q is in J, a contradiction.

The complexity bound follows since in each iteration we can check whether forward chaining adds a new atom in time $|H|n^t$ and there are at most $|\mathcal{S}|n^r$ iterations. □

The above process is similar of the use of the chase procedure to decide on uniform containment of database queries [Sag88]. Since we have access to EntMQ we can make sure that all clauses in the hypothesis of the algorithm are implied by the target function. (This essentially replaces the positive counter examples in the interpretations setting with EntMQ in the entailment setting.) Thus, the following lemma indicates that in the presence of EntMQ we can replace EQ by EntEQ.

Lemma 8. *Let $T \in \mathcal{H}_R(\mathcal{S}, =)$, $H \in \mathcal{H}_R(\mathcal{S}, =)$ and $T \models H$. Given a positive (clause) counter example $C \in \mathcal{H}_R(\mathcal{S}, =)$ such that $T \models C$ and $H \not\models C$ one can find a negative (interpretation) counter example I such that $I \not\models T$ and $I \models H$.*

Proof. This easily follows from the previous lemma since $I \not\models C$ and $T \models C$ implies $I \not\models T$. □

We therefore get that the class is learnable. The complexity of the algorithm is as in the interpretation setting (though a slightly more careful argument is needed).

Theorem 3. *The class $\mathcal{H}_R(\mathcal{S}, =)$ is learnable from entailment equivalence queries and entailment membership queries.*

As before we can get a learnability result for $\mathcal{H}_R(\mathcal{S})$; this time, however, we can use a hypothesis in $\mathcal{H}_R(\mathcal{S})$. Note that when learning $T \in \mathcal{H}_R(\mathcal{S})$, interpretation counter examples constructed by the model finding algorithm have a special structure. In particular, since $C \in \mathcal{H}_R(\mathcal{S})$ (in Lemma 8) every object in I (of Lemma 7) has a unique term associated with it (as generated from C). It follows that in the clauses generated in *rel-cands*() each variable has at most one defining equation. Therefore, the clauses can be "folded back" from the equational form into a range restricted form.

Theorem 4. *The class $\mathcal{H}_R(\mathcal{S})$ is learnable from entailment equivalence queries and entailment membership queries.*

6 The Implication Problem

Once expressions are learned as above one would want to use them as a knowledge base in a system, for example to infer properties implied by this knowledge. It is therefore useful if the implication problem is decidable and its complexity is bounded. It is easy to see that the model finding algorithm from Lemma 7 can be used to decide the implication problem.

Corollary 2. *The implication problem for $\mathcal{H}_R(\mathcal{S}, =)$ is decidable in time $O(|H| \cdot |\mathcal{S}| \cdot n^{t+r})$.*

We note that similar problems have been studied in database theory; checking whether $I \models H$ corresponds to query evaluation, and checking whether $H \models C$ corresponds to uniform containment [Sag88]. Completeness results for these problems parametrised by the number of variables in a clause follow from [PY97].

7 Discussion

We have shown that weakly range restricted Horn expressions are learnable from equivalence and membership queries, both for learning from interpretations and for learning from entailment. In the special case where the target expression is range restricted, we can use range restricted expressions as the hypothesis language for learning from entailment. For learning from interpretations hypotheses are weakly range restricted. Our results use flattening and unflattening of examples and clauses and a model finding procedure for this class.

The learning algorithm is similar to the algorithm for learning from entailment in the propositional case [FP93] as well as several previous ILP algorithms. In fact, the construction in Lemma 7 corresponds to elaboration in [SB86] and saturation in [Rou92], and flattening has been used before in [Rou92]. The pairing procedure from [Kha98] is similar to LGG computation [Plo70] used in many systems. In addition the dropping of non-legal literals is similar to what is done in [Rou92]. As we have shown a combination of these steps is formally justified in that it leads to convergence for range restricted Horn expressions.

Previous work in [Ari97,RT98,RS98] pursued similar problems in the context of learning from entailment. These works use oracles that are stronger than the ones used here in that they provide information on the syntax of the learned expression (using the order on atoms for acyclic expressions or otherwise information on subsumption rather than implication). On the other hand they derive complexity bounds that are lower than the ones here, in particular avoiding the exponential dependence on the number of terms in a clause. This is partly due to use of strong oracles and partly due to the the fact that different subclasses of Horn expressions are studied.

A natural question from the discussion above is whether the exponential dependence on the number of terms can be avoided without using the additional oracles. On the other hand, relaxing the requirement that clauses are range restricted is also of interest since many standard logic programs use recursive patterns that do not conform to it. Finally, in the model inference problem [Sha91] the learner is trying to acquire information about a model rather than a formula. In contrast with the scenario here the domain and mapping of function symbols are fixed and hence the nature of the problem is different. More work is needed to clarify these issues.

References

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

[AK95] D. Angluin and M. Kharitonov. When won't membership queries help? *Journal of Computer and System Sciences*, 50:336–355, 1995.

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

[Ari97] H. Arimura. Learning acyclic first-order Horn sentences from entailment. In *Proceedings of the International Conference on Algorithmic Learning Theory*. Springer-verlag, 1997. LNAI 1316.

[CK90] C. Chang and J. Keisler. *Model Theory*. Elsevier, Amsterdam, Holland, 1990.

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

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

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

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

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

[FP93] M. Frazier and L. Pitt. Learning from entailment: An application to propositional Horn sentences. In *Proceedings of the International Conference on Machine Learning*, pages 120–127, Amherst, MA, 1993. Morgan Kaufmann.

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

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

[Kha98] R. Khardon. Learning function free Horn expressions. Technical Report ECS-LFCS-98-394, Laboratory for Foundations of Computer Science, Edinburgh University, 1998. A preliminary version of this paper appeared in COLT 1998.

[Lit88] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

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

[MB92] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.

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

[MF92] S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.

[Min88] J. Minker, editor. *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1988.

[NCDW97] S. Nienhuys-Cheng and R. De Wolf. *Foundations of Inductive Logic Programming*. Springer-verlag, 1997. LNAI 1228.

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

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

[Rou92] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, 1992.

[RS98] K. Rao and A. Sattar. Learning from entailment of logic programs with local variables. In *Proceedings of the International Conference on Algorithmic Learning Theory*. Springer-verlag, 1998. LNAI 1501.

[RT97] C. Reddy and P. Tadepalli. Learning Horn definitions with equivalence and membership queries. In *International Workshop on Inductive Logic Programming*, pages 243–255, Prague, Czech Republic, 1997. Springer. LNAI 1297.

[RT98] C. Reddy and P. Tadepalli. Learning first order acyclic Horn programs from entailment. In *International Conference on Inductive Logic Programming*, pages 23–37, Madison, WI, 1998. Springer. LNAI 1446.

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

[Sag88] Y. Sagiv. Optimizing datalog programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, 1988.

[SB86] C. Sammut and R. Banerji. Learning concepts by asking questions. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning : An AI Approach, Volume II*. Morgan Kaufman, 1986.

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

[Sha91] E. Shapiro. Inductive inference of theories from facts. In J. Lassez and G. Plotkin, editors, *Computational Logic*, pages 199–254. MIT Press, 1991.

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

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