Learning Range Restricted 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

We study the learnability of first order Horn expressions from equivalence and membership queries. We show that the class of expressions where every term in the consequent of a 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 result for learning from interpretations is derived by exhibiting a reduction to the function free case (which was previously shown to be learnable). The reduction uses flattening of clauses - replacing a function symbol with a predicate symbol of arity larger by one - 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.

We also motivate the choice of restriction by showing that for such expressions implication is decidable. Hence, the learned expressions can be used as a knowledge base in a system in a useful way.

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

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 [Sha83, DRB92]. More recently Reddy and Tadepalli [RT97, RT98] 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 previous work [Kha98] we have shown 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 in the antecedent of the clause.

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 [DR97] and were both studied before. We present algorithms for learning range restricted Horn expressions in both settings. We also motivate the choice of restriction by showing that for such expressions implication is decidable. Hence, the 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. The reduction uses flattening of clauses [Rou92] - replacing a function symbol with a predicate symbol of arity larger by one - and an axiomatisation of the functionality of the new predicates. The hypothesis language allows for use of equalities in the Horn clauses, under a certain syntactic restriction generalising the the range restricted form. 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. Interestingly, the reduction uses learning from interpretations in a particular way that allows us to use range restricted expressions as the hypothesis language.

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. We will make use of further restrictions:

Definition 2.1 (definite clauses) *A clause is definite if it includes precisely one positive literal.*

Definition 2.2 (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 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.

Definition 2.3 $(\mathcal{H}(\mathcal{S}))$ *Let $\mathcal{S}$ be a signature. Then $\mathcal{H}(\mathcal{S})$ is the set of all Horn expressions over $\mathcal{S}$ in which all clauses are definite and range restricted.*

Our hypothesis language will allow a restricted use of equalities. To motivate the next definition consider the following clause and its equivalent form

$$\begin{aligned} (p_1(x_1, f_1(x_2)) \wedge p_2(f_2()) &\rightarrow p_1(x_1, f_2())) \\ ((z_1 = f_1(x_2)) \wedge (z_2 = f_2()) \wedge p_1(x_1, z_1) \wedge p_2(z_2) &\rightarrow p_1(x_1, z_2)) \end{aligned}$$

[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.

where we have replaced each term t_i with a new variable z_i but have added an equality $(z_i = t_i)$ between them. In this way each z_i has one defining equation, and we may think of the variables involved in the equation as its ancestors (e.g. x_2 is an ancestor of z_1). Constructed in this way all variables that appear in equational literals are ancestors of some variable in the original literals. We will consider the case where z_i may have more than one such equation as in

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

but where the variables in equations are still ancestors in this sense. These ideas are formalised in the following definitions.

Definition 2.4 (root variables, legal ancestor) *Let C be a definite Horn clause with equalities in its antecedent and where every non-equational literal includes only variables as terms, and every equational literal is of the form $(z_i = f_j(x_1, \ldots, x_n))$.*

- *The variables appearing in non-equational literals in the antecedent are called* root variables.
- *Root variables are* legal ancestors.
- *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 2.5 (weakly range restricted clauses) *A definite Horn clause with equalities in its antecedent (in the above form) is called weakly range restricted if every variable that appears in its consequent or in equational literals is a legal ancestor.*

As exemplified above, range restricted clauses have a semantically equivalent form (according to the standard semantics discussed below) which is weakly range restricted.

Definition 2.6 ($\mathcal{H}(\mathcal{S}, =)$) *Let $\mathcal{S}$ be a signature. Then $\mathcal{H}(\mathcal{S}, =)$ is the set of all Horn expressions over $\mathcal{S}$ in which all clauses are definite and weakly range restricted.*

2.2 Examples

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 I of the predicates and function symbols in $\mathcal{S}$ [Llo87]. 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 values 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. Let $str(\mathcal{S})$ be the set of interpretations for the signature $\mathcal{S}$

Examples of this form have been used in [Hau89, RT97, Kha98] and are motivated by the scenario of acting in structural domains (e.g. [Kha96, RTR96]). They are also used in the non-monotonic form of ILP [DRD94]. In structural domains, domain elements are objects in the world and an instantiation describes properties and relations of objects. We therefore refer to domain elements as *objects*.

2.3 Semantics

We use the standard semantics [CK90, Llo87]. Some basic notions are defined here to introduce the notation. Let I be an interpretation, T a set of terms (closed under subterms), X the set of variables appearing in T, and θ a mapping of the variables in X to objects in I. The *term assignment* of terms with respect to I and θ is defined inductively according to the depth of the term. For terms of depth 0: a variable $x \in X$ is mapped to $\theta(x)$, and a constant $f() \in F$ is mapped to the object it corresponds to in I. For a term $f(t_1, \ldots, t_n)$ of depth $i+1$, let $a_1, \ldots, a_n$ be the term assignments of $t_1, \ldots, t_n$; then $f(t_1, \ldots, t_n)$ is mapped to the object $f(a_1, \ldots, a_n)$ corresponds to in I.

Let $l(t_1, \ldots, t_n)$ be a literal, I an interpretation and θ a mapping of the variables in X to objects in I. The *ground literal* $l\theta = l(t_1\theta, \ldots, t_n\theta)$ is obtained from l by substituting variables in it according to θ. A ground positive literal $l(t_1, \ldots, t_n)$ is true in I with respect to θ if and only if $l(a_1, \ldots, a_n)$ is true in I where $a_1, \ldots, a_n$ are the term assignments of $t_1, \ldots, t_n$. A ground negative literal is true in I if and only if its negation is not.

A clause $C \in \mathcal{H}(\mathcal{S})$ is true in an interpretation I if for every substitution θ at least one of the literals in $C\theta$ is true in I. A Horn clause is therefore not true (falsified) in I if there is a substitution that simultaneously satisfies the antecedent and falsifies the consequent. An expression $T \in \mathcal{H}(\mathcal{S})$ 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 $T_1, T_2 \in \mathcal{H}(\mathcal{S})$ 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.

2.4 The Learning Model

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 function. 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).[2]

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

[2]Clearly, if $\mathcal{H}'$ is different from $\mathcal{H}$, the above definition still makes sense if we have a clear notion of what $I \models H$ means for $H \in \mathcal{H}'$.

the definition above). For complexity we measure the running time of the algorithm and the number of times it makes queries to EQ and MQ. It is well known [Lit88, Ang88] that learnability in this model implies pac-learnability [Val84].

3 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, $\mathcal{H}(\mathcal{S}')$ the set of 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 $T \in \mathcal{H}(\mathcal{S})$ to $T' \in \mathcal{H}(\mathcal{S}')$, 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 $T \in \mathcal{H}(\mathcal{S})$ will be constructed out of a learner for $T' \in \mathcal{H}(\mathcal{S}')$ (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 transfered to the membership oracle for T.

The Representation Transformation: Let $T \in \mathcal{H}(\mathcal{S})$ be a Horn expression, then the expression $\mathit{flat}(T) \in \mathcal{H}(\mathcal{S}')$ is formed by unfolding terms in C bottom-up and replacing them with variables. Formally, $\mathit{flat}(T)$ is defined by transforming 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 the clause by replacing all occurrences of this term with a new variable z, and adding a new literal $f_p(x_1, \ldots, x_n, z)$ 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

$$(f_{1_p}(x_2, z_1) \wedge p_1(x_1, z_1) \wedge p_2(f_2()) \rightarrow p_1(x_1, f_2())).$$

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

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

In similarity with the definitions of $\mathcal{H}(\mathcal{S}, =)$ we say that a variable in $\mathit{flat}(C)$ is a *root variable* if it appears in a literal $p(\ldots)$ in the antecedent for $p \in P$. For every literal $f_p(x_1, \ldots, x_n, z)$ in the antecedent we say that $x_1, \ldots, x_n$ are ancestors of z, and take the transitive extension of the ancestor relation (in the above example x_2 is an ancestor of z_2).

Lemma 3.1 *For $C \in \mathcal{H}(\mathcal{S})$, every variable in flat$(C)$ is either a root variable or an ancestor of a root variable.*

Proof: By construction this holds for variables in $f_p()$ literals. It holds for variables in the consequent since C is range restricted. ∎

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

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

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. Finally, *ax-flat*$(T) = \phi_F \wedge$ *flat*(T)

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.

Let $C \in \mathcal{H}(\mathcal{S})$ be a clause, $I \in str(\mathcal{S})$, and θ a mapping of variables of C to objects of I. The *term-assignment extension* of θ is a mapping θ' of variables in *flat*(C) to objects of *flat*(I) that is defined inductively as follows. Initialise θ' to be θ. In the process of flattening a clause defined above, each time a term $f(x_1, \ldots, x_n)$ is replaced with a variable z and the atom $f_p(x_1, \ldots, x_n, z)$ added, θ' maps z to the object that $f(x_1\theta', \ldots, x_n\theta')$ corresponds to in I. Hence it simply maps a variable to the object corresponding to the term assignment of the term that generated it. Since the term assignment is unique θ' is well defined.

Lemma 3.2 *Let $C \in \mathcal{H}(\mathcal{S})$ be a clause, $I \in str(\mathcal{S})$, θ a mapping of variables of C to objects of I, and θ' the term-assignment extension of θ. Then*
(1) For all $f_p \in F_p$, every literal $f_p(x_{i_1}, \ldots, x_{i_{n+1}})$ in the antecedent of flat(C) is true in flat(I) with respect to θ'.
(2) For all $p \in P$ and every atom $l = p(t_1, \ldots, t_n)$ in C and the corresponding atom $l' = p(x_{i_1}, \ldots, x_{i_n})$ in flat(C): l is true in I with respect to θ if and only if l' is true in flat(I) with respect to θ'.

Proof: Each literal $f_p(\ldots)$ is generated by some term $f(\ldots)$ in C. For a constant $f()$, the literal is $f_p(x)$, and θ' maps x to the object a to which $f()$ corresponds in I. By the construction of *flat*(I), $f_p(a)$ is true in *flat*(I). For a term $f(t_1, \ldots, t_n)$ of depth i the corresponding atom is $f_p(x_{i_1}, \ldots, x_{i_n}, z)$, where each x_{i_j} is mapped by θ' to a_j, the term assignment of t_j in I with respect to θ. Furthermore, z is mapped by θ' to the object $f(a_1, \ldots, a_n)$ corresponds to in I; let this object be a. Again , by the construction of *flat*(I), $f_p(a_1, \ldots, a_n, a)$ is true in *flat*(I).

For (2) we observe as above that θ' maps each x_{i_j} to the term assignment a_j of t_j in I with respect to θ. Since predicates in P have the same extension in I and *flat*(I) we obtain the equivalence. ■

Lemma 3.3 *For all* $T \in \mathcal{H}(\mathcal{S})$ *and for all* $I \in str(\mathcal{S})$
(1) $flat(I) \models \phi_F$
(2) $I \models T$ *if and only if* $flat(I) \models flat(T)$
(3) $I \models T$ *if and only if* $flat(I) \models$ *ax-flat*(T)

Proof: Since each constant and each ground term of depth 1 are mapped to precisely one domain element in I, (1) is true by the construction of *flat*(I). Clearly, given (1), part (3) follows from (2).

For (2) , assume first that $I \not\models T$. Hence there is a substitution θ such that $C\theta$ is not true in I for some clause C in T. We claim that $flat(I) \not\models flat(C)$ with respect to θ', the term-assignment extension of θ. Therefore, $flat(I) \not\models flat(T)$.

For the claim observe that by Lemma 3.2 all literals in the antecedent of *flat*(C) involving predicates in F_p are true in I, θ', and that all atoms involving predicates in P have the same truth value as in C with respect to I, θ. Therefore both antecedent and consequent of *flat*(C) have the same truth value as of C in I, θ and the clause is falsified.

Assume on the other hand that $flat(I) \not\models flat(C)$. Hence there is a substitution θ' such that the antecedent of *flat*(C) is true in *flat*(I) with respect to θ'. Now, by construction, the extension of each f_p predicate in *flat*(I) is functional. Hence, since all f_p predicates are true with respect to θ', it follows that θ' is a term-assignment extension of some substitution θ to the variables of C. As above it follows that $I \models C\theta$ if and only if $flat(I) \models flat(C)\theta'$ thus proving the result. ■

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 predicates. Such interpretations are not output by the algorithm of [Kha98] when learning $\mathcal{H}(\mathcal{S}')$ 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 all ground instances of the existence clauses false in J are made true by adding a positive atom whose last term is $*$. More formally, let *non-complete*(J) have the additional object $*$ but the same extension as J. For any set $a_1, \ldots, a_n$ of domain elements of *non-complete*(J) such that $\exists z f_p(a_1, \ldots, a_n, z)$ is not true in *non-complete*(J), put $f_p(a_1, \ldots, a_n, *)$ in a set of "additional atoms". Then *complete*(J) has the additional element $*$ and the extension of predicates is defined by the union of positive atoms true in J and the additional atoms defined above.

For any $J \in str(\mathcal{S}')$ such that $J \models \mathcal{A}_{unique}$, the interpretation J is thus transformed into *unflat*(*complete*(J)).

Lemma 3.4 *For all $T \in \mathcal{H}(\mathcal{S})$ and for all $J \in str(\mathcal{S}')$ such that $J \models \mathcal{A}_{unique}$:*
$J \models flat(T)$ if and only if $complete(J) \models ax\text{-}flat(T)$.

Proof: Assume first that $J \not\models flat(T)$. Thus there is a substitution θ such that $J \not\models C\theta$ for some C in $flat(T)$. The same substitution can be used to show that $complete(J) \not\models flat(T)$ and hence also $complete(J) \not\models ax\text{-}flat(T)$.

Assume on the other hand that $complete(J) \not\models ax\text{-}flat(T)$. Then, since $complete(J) \models \phi_F$ it follows that $complete(J) \not\models flat(T)$. Let C be a clause in T and θ a substitution such that $complete(J) \not\models flat(C)\theta$, and the antecedent of C is satisfied by θ in $complete(J)$.

Since C is range restricted, it follows that no variable in $flat(C)$ is mapped to the new object $*$ by θ. To see that observe that by Lemma 3.1 every variable in $flat(C)$ is an ancestor of some root variable in $flat(C)$. Now, by construction if x is an ancestor of y and x is mapped to $*$ by θ then, since the antecedent is satisfied by θ, y is also mapped to $*$ by θ (since the uniqueness axiom is satisfied in $complete(J)$). It follows that some root variable x is mapped to $*$ by θ. Therefore, by the construction, some literal $p(\ldots, x, \ldots)$ (where $p \in P$) in the antecedent of $flat(C)$ is made false by θ. Hence no variable is mapped to $*$.

Therefore θ can be used as a substitution in J and $J \not\models flat(C)\theta$, implying $J \not\models flat(T)$. ■

Lemma 3.5 *For all $T \in \mathcal{H}(\mathcal{S})$ and for all $J \in str(\mathcal{S}')$ such that $J \models \mathcal{A}_{unique}$:*
$J \models flat(T)$ if and only if $unflat(complete(J)) \models T$.

Proof: Note that $unflat(complete(J)) \in str(\mathcal{S})$ and $flat(unflat(complete(J))) = complete(J)$. Hence, by Lemma 3.3(3), $unflat(complete(J)) \models T$ if and only if $complete(J) \models ax\text{-}flat(T)$. The lemma thus follows from Lemma 3.4. ■

For a clause $C \in \mathcal{H}(\mathcal{S})$, 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 sub-terms. For example, $(p(x, f_1(x), f_2(f_1(x)), f_3()) \rightarrow q(f_1(x)))$ has 4 distinct terms $x, f_3(), f_1(x), f_2(f_1(x))$.

Theorem 3.6 *Let $\mathcal{S} = (P, F)$ be a signature, and let r be the maximal arity of predicates and function symbols in $\mathcal{S}$.*

- *$\mathcal{H}(\mathcal{S})$ is learnable from equivalence and membership queries with hypothesis in $\mathcal{H}(\mathcal{S}')$.*
- *For $T \in \mathcal{H}(\mathcal{S})$ with m clauses and at most k distinct terms in a clause, the algorithm is polynomial in $|P| + |F|$ and m and exponential in r, k.*

Proof: The theorem follows from properties of pwm-reductions [AK95] and the result in [Kha98] showing that $\mathcal{H}(\mathcal{S}')$ is learnable.

Essentially the idea is that when learning $T \in \mathcal{H}(\mathcal{S})$ we will run the algorithm *A2* from [Kha98] to learn the expression $flat(T) \in \mathcal{H}(\mathcal{S}')$. When *A2* presents $H \in \mathcal{H}(\mathcal{S}')$ to an equivalence query we interpret this by saying that $I \in str(\mathcal{S})$ is a models 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 3.3(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 3.5 guarantees that the answer is correct. By Corollary 11 of [Kha98] we get that *A2* will find an expression equivalent to *flat*(T).

It remains to observe that each distinct term in a clause $C \in \mathcal{H}(\mathcal{S})$ is mapped to a variable in *flat*(C). The complexity bound follows from [Kha98]. ■

4 Modifying the Hypothesis Language

The previous theorem produces a hypothesis in $\mathcal{H}(\mathcal{S}')$ while the target function is in $\mathcal{H}(\mathcal{S})$. Modifying the algorithm so as to work with expressions in $\mathcal{H}(\mathcal{S})$ will clearly be of interest. Failing to do this we show how one can use a hypothesis in $\mathcal{H}(\mathcal{S},=)$ where the same signature as in $\mathcal{H}(\mathcal{S})$ is used.

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-cands*(J) where *rel-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 replacing every literal of the form $f_p(x_1, \ldots, x_n, x_{n+1})$ 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) may not be in $\mathcal{H}(\mathcal{S},=)$ since some of the variables in its equality literals may not be legal ancestors (cf. Definition 2.4).

Lemma 4.1 *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$(I) \models C$.

Proof: Note that C and *unflat*(C) have the same variables and *flat*(I) and I have the same domain. It follows by the construction of *flat*(I) that for any θ mapping these variables to domain elements, $(x_{n+1} = f(x_1, \ldots, x_n))\theta$ is true in I if and only if $f_p(x_1, \ldots, x_n, x_{n+1})\theta$ is true in *flat*(I). The result follows since the truth values of predicates in P is not changed by this transformation. ■

In fact the lemma does not depend on C being generated by the *rel-cands*() operation. However, when applied in this way we observe 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 $\mathcal{H}(\mathcal{S},=)$. We first define legal objects of interpretations in similarity with legal ancestors in clauses. Let $J \in str(\mathcal{S}')$, and let D be the domain of J. Here again *legal objects* are defined inductively. If $p(a_1, \ldots, a_n)$ is true in J, for $p \in P$ then $a_1, \ldots, a_n$ are legal objects. If

$f_p(a_1, \ldots, a_n, a_{n+1})$ is true in J where $f_p \in F_p$, and a_{n+1} is a legal object then $a_1, \ldots, a_n$ are legal objects.

The idea is that non-legal objects are somehow "not connected" to the important objects of the interpretation (corresponding to root variables in clauses) and hence the atoms referring to them can be dropped. 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 4.2 *Let $T \in \mathcal{H}(\mathcal{S})$ 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: Assume first that $unflat(complete(J)) \not\models T$. By Lemma 3.4 and Lemma 3.5 we have that $complete(J) \not\models flat(T)$. Therefore, there is a θ such that $complete(J) \not\models flat(C)\theta$ for some C in T.

By the definition of non-legal objects and the construction of $complete(J)$, if a is non-legal (or the object $*$) and $b = f(\ldots, a, \ldots)$ in $complete(J)$, then b is either a non-legal object or the object $*$ in $complete(J)$.

Recall that by Lemma 3.1 every variable in $flat(C)$ is an ancestor of some root variable in $flat(C)$. Therefore, since the antecedent of $flat(C)$ is satisfied by θ in $complete(J)$ and the uniqueness axiom holds in $complete(J)$, we get that if some variable in $flat(C)$ is mapped to a non-legal object in $complete(J)$ then some root variable in $flat(C)$ is mapped either to a non-legal object or to $*$. It follows that, in such a case, some literal $p(\ldots)$ (where $p \in P$) in the antecedent of $flat(C)$ is made false by θ, and $complete(J) \models flat(C)\theta$. Hence no variable is mapped to a non-legal object (or to $*$) in J by θ. Therefore θ can be used in $complete(J_{|\{D \setminus a\}})$ without altering the truth value of $flat(C)$ implying that $complete(J_{|\{D \setminus a\}}) \not\models flat(C)$.

For the other direction, the same argument shows that if $complete(J_{|\{D \setminus a\}}) \not\models flat(C)$ then the falsifying substitution does not use the object $*$. Hence, it can be used in J without altering the truth value of $flat(C)$ and the result follows. ■

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 4.2 we get that all objects in all interpretations in S are legal objects. This in turn implies that the hypothesis is in $\mathcal{H}(\mathcal{S}, =)$.

Corollary 4.3 *The class $\mathcal{H}(\mathcal{S})$ is learnable from equivalence and membership queries with hypothesis in $\mathcal{H}(\mathcal{S}, =)$.*

5 Learning from Entailment

In this model examples are clauses in the underlying language $\mathcal{H}(\mathcal{S})$ [FP93]. An example $C \in \mathcal{H}(\mathcal{S})$ is positive for $T \in \mathcal{H}(\mathcal{S})$ 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). The following lemma indicates that we can replace MQ by EntMQ for clauses in $\mathcal{H}(\mathcal{S}, =)$.

Lemma 5.1 *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, recall that we can rewrite range restricted clauses in equational form as illustrated in Section 2. The equational form of $C \in \mathcal{H}(\mathcal{S})$ can be obtained by computing $unflat(flat(C))$. Let R be a clause in T in its equational form and θ a substitution such that $I \not\models R\theta$.

Let γ be the (reverse) substitution that is used when generating $rel\text{-}cands(J)$. Since the antecedent of R is satisfied by θ in J we get that $ant(R)\theta\gamma \subseteq ant(unflat(C))$ 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 $rel\text{-}cands(R)$ all consequents are considered, we get that for some $C' \in rel\text{-}cands(J)$, $R\theta\gamma \subseteq unflat(C')$.[3] We therefore get that $T \models R \models R\theta\gamma \models unflat(C')$. ■

We next prove a lemma that is instrumental in the decidability result as well.

Lemma 5.2 *Given $H \in \mathcal{H}(\mathcal{S}, =)$ and a clause $C \in \mathcal{H}(\mathcal{S}, =)$ such that $H \not\models C$, one can find an interpretation $I \in str(\mathcal{S})$ such that $I \models H$ and $I \not\models C$.*

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 $I_0 \in str(\mathcal{S})$ as follows. First, introduce a unique domain element for each term in C and then collapse elements if their terms are equated in the antecedent of C; let this domain be D. 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. To make I_0 into an interpretation, (as in the *complete*() construction) add another domain element $*$ and map each term $f(a_1, \ldots, a_n)$ that has not yet been assigned to $*$.

[3] In other words R, θ-subsumes $unflat(C')$ [Plo70].

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 $I \not\models C$.

By the restriction on $\mathcal{H}(\mathcal{S}, =)$, all variables that appear in clauses in H are ancestors of at least one root variable. It follows that for any substitution in I_0 for which any of the variables in a clause is mapped to $*$ the antecedent is falsified. 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 $I_{0|D}$ into J.

Finally, assume that p is in I. Since its forward chaining does not touch 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 above process is similar to 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 5.3 *Let $T \in \mathcal{H}(\mathcal{S})$, $H \in \mathcal{H}(\mathcal{S}, =)$ and $T \models H$. Given a positive (clause) counter example $C \in \mathcal{H}(\mathcal{S})$ 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 $I \not\models T$. ■

In fact the above construction shows that interpretation counter examples found in this way have a special structure. In particular, since $C \in \mathcal{H}(\mathcal{S})$ (in Lemma 5.3) every object in I (of Lemma 5.2) has a unique term associated with it (as generated from C). It follows that in the clauses generated in *rel-cands*(I) each variable has at most one defining equation. Therefore, the clauses can be "folded back" from the equational form into a range restricted form. This implies:

Corollary 5.4 *The class $\mathcal{H}(\mathcal{S})$ is learnable from entailment equivalence queries and entailment membership queries with hypothesis in $\mathcal{H}(\mathcal{S})$.*

6 Decidability

It makes sense to concentrate on a subset of the language for which implication is decidable. In such a case the learned expressions can be used as the knowledge base of the learner. Using the techniques developed for learning from clauses we can show that this holds for $\mathcal{H}(\mathcal{S},=)$. The complexity of this algorithm depends directly on the complexity of the forward chaining process; while we do not discuss this here there is a clear relationship to query evaluation in database theory (see discussion in [Sag88]) and more refined complexity results in the line of [PY97] can probably be derived.

Theorem 6.1 *Implication is decidable for expressions in* $\mathcal{H}(\mathcal{S},=)$.

Proof: To decide whether $H \models C$ where $H, C \in \mathcal{H}(\mathcal{S},=)$ construct the interpretation I as in Lemma 5.2. Then check if C is falsified by I (in which case $H \not\models C$) or not ($H \models C$). ■

7 Discussion

While we have mostly presented various transformations between learning problems and examples, the resulting algorithm can be specified using the signature $\mathcal{S}$ alone. Moreover, for learning from entailment, pairing and other operations used in [Kha98] can be performed directly on clauses (or their antecedents) without resorting to the use of interpretations.

The resulting 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 5.2 corresponds to elaboration in [SB86] and saturation in [Rou92], and flattening has been used before in [Rou92]. Our pairing procedure and the various minimisations of objects correspond to the LGG computation or the heuristic search used, but provide a more refined way to control the complexity of the process. 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.

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.

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

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

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

[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 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. Lecture notes in Artificial Intelligence, vol. 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.

[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.

[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.