Learning Horn Definitions with Equivalence and Membership Queries

Chandra Reddy & Prasad Tadepalli

Dearborn 303, Department of Computer Science
Oregon State University, Corvallis, OR-97331. USA
Phone: +1 541 753 2770; Fax: +1 541 737 3014
{reddyc,tadepalli}@cs.orst.edu

Abstract. A Horn definition is a set of Horn clauses with the same head literal. In this paper, we consider learning non-recursive, function-free first-order Horn definitions. We show that this class is exactly learnable from equivalence and membership queries. It follows then that this class is PAC learnable using examples and membership queries. Our results have been shown to be applicable to learning efficient goal-decomposition rules in planning domains.

1 Introduction

In this paper, we consider learning Horn definitions — multiple Horn clauses with a single head — in the exact learning model [Angluin, 1988]. Learning Horn definitions is a fundamental problem both in the Inductive Logic Programming (ILP) literature as well as the Computational Learning Theory literature. Since it is NP-hard to test membership in this concept class, it immediately follows that even non-recursive function-free Horn definitions are hard to learn from examples alone [Schapire, 1990]. Using only equivalence queries, single non-recursive Prolog clauses are learnable [Cohen, 1995a, Džeroski, Muggleton, & Russell, 1992] with restrictions such as determinacy and bounded arity. Restricted versions of single recursive clauses are also shown to be learnable [Cohen, 1995b]. However, learning multiple clauses or even slightly more general versions of either recursive or non-recursive clauses is shown to be hard without further help [Cohen, 1995c]. Page [1993] has shown that non-recursive Horn defintions with predicates having fixed arity and with the restriction that only the variables that occur in the head literal of a clause appear in the body of the clause are learnable using equivalence and subset queries. In this paper, we examine the learnability of a more general class.

In particular, we show that first-order non-recursive function-free Horn definitions are exactly learnable from membership and equivalence queries with no other restrictions. Our approach is similar to Angluin's algorithm for learning Propositional Monotone DNF [Angluin, 1988]. The algorithm exploits the fact that there is at most one positive literal in a Horn clause, which makes it possible to show that any clause which is implied by the target must be subsumed by one of the clauses in the target. Our membership queries, in effect, check whether a hypothesis clause is subsumed by a target clause. Since it is NP-complete to decide subsumption, for the teacher to answer the membership queries efficiently, in practice, the hypothesis language should be restricted so that deciding subsumption is efficient. Kietz and Lübbe [1994] show that the restrictions to determinate clauses or to constant-locailty clauses make deciding subsumption a polynomial-time problem.

Our research is motivated by a practical application that involves learning goal-decomposition rules (d-rules) in planning. We show that this problem can be reduced to one of learning Horn definitions, which enabled us to apply our learning algorithm for Horn definitions to efficiently learn d-rules.

The rest of the paper is organized as follows: Section 2 presents some formal preliminaries about Horn definitions. Section 3 describes the learning problem, proves some properties of Horn definitions, describes the learning algorithm and proves its correctness. Section 4 employs this result to show that goal-decomposition rules are learnable. Section 5 concludes the paper by mentioning some implications of this result.

2 Preliminaries

In this section, we define and describe the terms we use in the rest of the paper, omitting some of the standard terminology and notation of logic (as given in standard books such as [Lloyd, 1987]).

Definition 1. A **Horn clause** is a finite (multi)set of literals that contains at most one positive literal. It is treated as a disjunction of the literals in the set with universal quantification over all the variables. The positive literal, if it exists, is called the *head* of the clause. A Horn clause is **function-free** if it does not contain any function symbols of arity 1 or more.[1] A Horn clause is **non-recursive** if the head of the Horn clause occurs only as a positive literal.

Definition 2. A **function-free Horn definition** is a set of function-free Horn clauses with all clauses having the same head literal. It is **non-recursive** if the head literal does not occur as a negative literal in any clause in the definition.

Definition 3. A clause D **subsumes** a clause E if there exists a substitution θ such that $D\theta \subseteq E$.[2] We denote this as $D \succeq E$, and read it as D subsumes E or as D is more general than E.

Definition 4. If D and E are clauses such that $D \succeq E$, then a literal l in a clause E is relevant (irrelevant) w.r.t the clause D, if $D \not\succeq E - \{l\}$ ($D \succeq E - \{l\}$, respectively).

[1] Note that this definition of a function-free Horn clause allows constants—functions with arity-0.

[2] Note that when we say $D \subseteq E$, we mean that all the *literals* in D are also in E—but not that the extension of D is in the extension of E. Also, $|D|$ is the number of literals in D, and not the size of the extension.

Definition 5. If C and D are two clauses such that $D \succeq E$, then a **condensation** of E w.r.t. D is a clause E' such that $E' \subseteq E$, $D \succeq E'$, and for any $l \in E'$, $D \not\succeq E' - \{l\}$.

For example, if $D = \{\neg l_1(x), l_2(y)\}$ and $E = \{\neg l_1(a), l_2(b), l_2(c), l_3(c)\}$, then both $\{\neg l_1(a), l_2(b)\}$ and $\{\neg l_1(a), l_2(c)\}$ are condensations of E w.r.t. D.

Definition 6. **Least general generalization** of a set of clauses S over $\succeq$ is a clause D such that (1) for every E in S, $D \succeq E$, and (2) if there exists a clause F such that for every E in S, $F \succeq E$, then $F \succeq D$.

The definitions 3 and 6 are due to Plotkin[1970]. For a proof of existence of least general generalization see [Plotkin, 1970] and [Nienhuys-Cheng & de Wolf, 1996].

We follow the description in [Muggleton & Feng, 1990] of Plotkin's algorithm to find the least general generalizations (lgg) of a set of clauses. The lgg of two clauses C_1 and C_2 is $\bigcup_{p_1 \in C_1, p_2 \in C_2} lgg(p_1, p_2)$. The lgg of two predicates $p(a_1, a_2 \ldots a_n)$ and $p(b_1, b_2 \ldots b_n)$ is $\{p(lgg(a_1, b_1), lgg(a_2, b_2) \ldots lgg(a_n, b_n))\}$; if the predicate symbols are not equal, their sign differs or their arity differs, then their lgg is $\{\}$, the empty set. The lgg of two terms a and b is itself, if they are the same; else, it is a variable v, where v stands for $lgg(a, b)$ throughout the computation of the lgg of the pair of clauses.

For example, let $C_1 = \{\neg l_1(a, b), \neg l_2(a, c), l_3(b)\}$ and $C_2 = \{\neg l_1(c, d), \neg l_1(b, a), \neg l_2(c, c), l_3(a)\}$. Then $lgg(C_1, C_2) = \{\neg l_1(x, y), \neg l_1(z, u), \neg l_2(x, c), l_3(u)\}$, where the variables x, y, z and u stand for the pairs $lgg(a, c)$, $lgg(b, d)$, $lgg(a, b)$ and $lgg(b, a)$, respectively.

Note that $|lgg(C_1, C_2)|$ can be equal to $|C_1| \times |C_2|$.

Lemma 7. *Let C_1, H_1 and H_2 be Horn clauses. Then $C_1 \succeq H_1$ and $C_1 \succeq H_2$ if and only if $C_1 \succeq lgg(H_1, H_2)$.*

Proof. The only-if part follows from the property (2) of the definition of least-general generalization. The if part follows from the transitive property of $\succeq$. □

We state the following fact explicitly, although it is straightforward, for it is useful later.

Proposition 8. *If $C_1 \succeq H_1$ then $C_1 \succeq H_2$ for any H_2 such that $H_1 \subseteq H_2$.*

3 Learning Horn Definitions

In this section, we first specify our learning problem. Next we describe the learning algorithm and then give the learnability result.

3.1 Learning Problem

Our learning problem is motivated by learning control knowledge for planning in structural domains. The following definitions reflect that motivation.

A *scene* is a conjunction of positive ground literals describing a set of objects.[3] We call the predicates that occur in scenes *base predicates*. We differentiate the base predicates from a special predicate called the *goal predicate*. An *instance* is a 2-tuple comprising a scene *scene* and a ground goal literal g, meaning that g is true in *scene*. We alternately write an instance as a clause $scene \rightarrow g$.

We consider the hypothesis space of Horn definitions for the goal predicate. An instance $\langle scene, g\rangle$ is in a hypothesis H iff the minimal model of H with respect to the literals in *scene* satisfies g. In other words, $\langle scene, g\rangle$ is an instance of H iff $H \models (scene \rightarrow g)$. Such an instance is a *positive example* of H. All other instances are *negative examples*.

Henceforth, Σ denotes the target concept in the hypothesis space.

Example 1. The following illustrates the above definitions in a toy version of an air-traffic control domain.
$\Sigma = \{$
`((plane-at ?p ?loc) (level L1 ?loc) (free-runway ?r)`
`(short-runway ?r) (can-land-short ?p))` $\rightarrow$ `(land-plane ?p)`,
`((plane-at ?p ?loc) (level L1 ?loc) (free-runway ?r)`
`(long-runway ?r))` $\rightarrow$ `(land-plane ?p)`
$\}$

The first clause in Σ gives the conditions under which a plane can land on short runways. The second clause is for long-runway landing. The following is a positive example of Σ (for the second clause):
`((plane-at P737 10) (lelvel L1 10) (free-runway R1) (long-runway R1)`
`(short-runway R2) (wind-speed high) (wind-dir south) (free-runway R2))`
$\rightarrow$ `(land-plane P737)`. □

Before stating the learning problem, we define the queries we will need [Angluin, 1988]. A *membership query* takes as input an instance x, and outputs *yes* if x is in Σ, and *no* otherwise. An *equivalence query* takes as input a set H of hypotheses, and outputs *yes* if the set of instances in H is the same as the set of instances in Σ; otherwise, returns a *counterexample* from $H \oplus \Sigma$ —i.e., an instance that is in one but not in the other.

The learning problem in the exact learning framework [Angluin, 1988] is the following. An algorithm *exactly learns* a concept in function-free non-recursive Horn definition representation in polynomial time using equivalence queries if and only if it runs in time polynomial in the size of Σ and in the size of the largest counterexample, and outputs a function-free non-recursive Horn definition that is extensionally equivalent to Σ.

3.2 Strong Compactness of Non-recursive Horn Definitions

In this section we describe a property of non-recursive Horn definitions, which is called *strong compactness* by Lassez, et al. [1988] and Page [1993], and relate this property to membership queries.

[3] We employ closed-world assumption and assume that all other literals are negative.

Strong compactness says that for non-recursive Horn definitions if we know that a clause is logically implied by a set of clauses Σ, then we can conclude that that clause is subsumed by a clause in Σ. The following lemma, in addition, says that the converse is true. This is useful to show later that each hypothesis in our algorithm is always a specialization of some target clause.

Lemma 9. *Let $\Sigma = \{C_1, C_2, \ldots, C_n\}$ be a set of horn clauses, where each C_i is $\{\neg c_{i1}, \ldots, \neg c_{ik_i}, p\}$, and $c_{i1}, \ldots, c_{ik_i}$ and p are first-order positive literals. Further let $H_1 = \{\neg h_1, \ldots, \neg h_m, q\}$ be a horn clause, where q and $h_1, \ldots, h_m$ are first-order positive literals. Also, suppose that the predicate symbols of p and q are distinct from those of c_{ij} and h_i. Then, $\Sigma \models H_1$ if and only if there exists a t, $1 \leq t \leq n$, such that $C_t \succeq H_1$. We call C_t the* **target clause** *of H_1, and H_1 the* **hypothesis clause** *of C_t.*

Proof. This is a direct consequence of the Subsumption theorem [Kowalski, 1970]. □

If a clause H_1 has variables, to determine $\Sigma \models H_1$ is to determine whether all instances in H_1 are also in Σ—which is the same as a subset query [Angluin, 1988]. However, by substituting each variable in H_1 by a unique constant, we can form an instance of H_1. We call this procedure Instantiate. Now, determining whether $\Sigma \models H_1$ is equivalent to asking whether $\Sigma \models \mathsf{Instantiate}(H_1)$. Asking whether $\Sigma \models \mathsf{Instantiate}(H_1)$ is the same as a membership query. In effect, this membership query simulates a subset query.

3.3 Learnability of Function-free Non-recursive Horn Definitions

Horn-learn is an algorithm to learn non-recursive function-free Horn definitions using equivalence and membership queries (Figure 1). Horn-learn makes use of Generalize algorithm. Generalize takes as input a Horn clause and generalizes it by eliminating literals from that Horn clause. It removes a literal from the Horn clause and checks whether the resultant Horn clause is overgeneral. It can do this by substituting each variable in the hypothesis clause with a unique constant and asking a membership query. If it is overgeneral the literal is not eliminated; otherwise, it is eliminated to form a new, more general Horn clause.

Horn-learn starts with hypothesis H that is initially empty. As long as H is not equivalent to the target concept C, the equivalence query returns an example e that is not included in H, and the algorithm modifies H to cover e. To include e in H, Horn-learn checks each Horn clause H_i of H whether generalizing H_i to cover e would not make the hypothesis overgeneral—i.e., a check whether $lgg(H_i, e)$ is in the concept. If so, concluding that it has found the right Horn clause H_i to include e in, Horn-learn further generalizes $h = lgg(H_i, e)$, by removing irrelevant literals using membership oracle (using Generalize), and replaces H_i in H by the new generalized h. If there is no such H_i, Horn-learn generalizes e and makes it a new Horn clause of H.

The generalization process of Generalize serves a critical purpose. Recall that the size of lgg grows as a product of the sizes of the hypotheses being generalized.

Unless the hypothesis size is limited, it can grow exponentially in the number of examples used to create the hypothesis. Lemma 10 and Lemma 11 together show that Generalize guarantees that the sizes of the hypothesis clauses are at most the sizes of their corresponding target clauses. Lemma 10 shows that Generalize does not overgeneralize in the process.

Horn-learn

Let Σ be the target concept.
$H := \{\}$ `/* empty hypothesis, initially */`
$m := 0$ `/* number of clauses in the hypothesis */`
while equivalence(H, Σ) is not true **and** e is a counterexample **do**
`/* fix the clause in` H `for the example` e `*/`
  if $(m > 0)$ **then**, Let $Hbe\{H_1, H_2, \ldots, H_m\}$
  found := false; $i := 1$
  while $(i \leq m)$ **and** found is false **do**
    $h := lgg(e, H_i)$
    if $\Sigma \models h$ **then** found := true;
        `/*` **Member?**(Instantiate(h)) `implements` $\Sigma \models h$`*/`
    else $i := i + 1$
  endwhile `/*` $i \leq m$ `*/`
  if found = false **then** $h := e$; $m := m + 1$;
`/* further generalize` h `*/`
  $H_i :=$ Generalize(h)
endwhile
return H

Generalize(h)

1. $h' := h$
2. **for each** literal l in h **do**
3. **if** $\Sigma \models h' - \{l\}$ **then** $h' := h' - \{l\}$
   `/*` Implemented by **Member?**(Instantiate($h' - \{l\}$)) `*/`
4. Return h'.

Fig. 1. Horn-learn: An algorithm to learn Horn definitions

Lemma 10. *If the argument h of* Generalize *is not overegeneral, then, at the end of* Generalize, *h' has a target Horn clause C_j—i.e., $C_j \succeq h'$. Moreover, h' in line 4 of* Generalize *is a condensation of h w.r.t. C_j.*

Proof. In the beginning of Generalize, h', which is the same as the argument h, is not overgeneral. h' is modified only when the modification still leaves the result inside Σ. That is, $\Sigma \models h'$. By Lemma 9, there exists a target Horn clause for h', say C_j, and $C_j \succeq h'$.

To show that h' in line 4 of Generalize is a condensation of h w.r.t. C_j, we need only to show that for any literal $l \in h'$, $C_j \not\succeq (h' - \{l\})$. Suppose that for some

$l \in h'$, $C_j \succeq (h' - \{l\})$. Let h'' be the value of h' when l is considered for removal in the loop of lines 2—3. Since $h' \subseteq h''$, by Proposition 8, $C_j \succeq (h'' - \{l\})$. From Lemma 9, $\Sigma \models (h'' - \{l\})$. In that case, l would have been removed by line 3 of Generalize. But, $l \in h'$, a contradiction. Therefore, for any literal $l \in h'$, $C_j \not\succeq (h' - \{l\})$. □

Lemma 11. *If h' is a condensation of h w.r.t. C_j, then $C_j\theta = h'$ for some substitution θ. Moreover, $|h'| \leq |C_j|$.*

Proof. Suppose h' is a condensation of h w.r.t. C_j. Then there exists a θ such that $C_j\theta \subseteq h'$. Suppose $C_j\theta \subset h'$. Then, for some $l \in h' - C_j\theta$, $C_j\theta \subseteq h' - \{l\}$. Hence, $C_j \succeq (h' - \{l\})$. This is a contradiction, since h' is a condensation w.r.t. C_j. Therefore, $C_j\theta = h'$. This implies that $|h'| = |C_j\theta| \leq |C_j|$. □

The following definition relates an example to a hypothesis clause and to a target clause.

Definition 12. If $C_1, C_2, \ldots, C_n$ are the Horn clauses in the target concept Σ, and $H_1, H_2, \ldots, H_m$ are the Horn clauses in the hypothesis H, then a *correct hypothesis Horn clause* in H for an example e is a Horn clause H_i such that for some $1 \leq j \leq n$, $C_j \succeq e$ and $C_j \succeq H_i$.

Lemma 13. *In* Horn-learn, *suppose that e is a counterexample returned by the equivalence query such that e is covered by Σ, but not by H. Then* Horn-learn *includes e in a correct hypothesis Horn clause in H for e if and only if one exists.*

Proof. First the only-if part. Horn-learn includes e in H_i of H if $\Sigma \models lgg(e, H_i)$. If $\Sigma \models lgg(e, H_i)$, then, by Lemma 9, $C_j \succeq lgg(e, H_i)$ for some C_j of C. Then, by Lemma 7, $C_j \succeq e$ and $C_j \succeq H_i$. Therefore, if Horn-learn includes e in H_i of H, then H_i is a correct hypothesis Horn clause for e.

Now, the if part of the claim. Let H_i be a correct hypothesis Horn clause for e in H such that no H_k such that $k < i$ is one. Then there exists a C_j of C such that $C_j \succeq e$ and $C_j \succeq H_i$. This implies, by Lemma 7, that $C_j \succeq lgg(e, H_i)$. By Lemma 9, $\Sigma \models lgg(e, H_i)$. Also, for $k < i$, $C_j \not\succeq H_k$, which implies $\Sigma \not\models lgg(e, H_k)$. Therefore, H_i is the first clause in the hypothesis H for which $\Sigma \models lgg(e, H_i)$. Then, by lines 7–13 in Fig. 1, e is included in H_i by assigning $lgg(e, H_i)$ to H_i (after further generalizing it using Generalize). □

Lemma 14. *Suppose that e is a counterexample such that e is covered by Σ, but not by H. Then* Horn-learn *adds a new Horn clause to H that includes e if and only if H does not already have a correct hypothesis Horn clause for e.*

Proof. By lines 11 and 12, and Lemma 13. □

Lemma 15. *The following are invariant conditions of* Horn-learn*:*
1. Every Horn clause H_i in the hypothesis H has a target clause;
2. Every Horn clause C_j in the target concept Σ has at most one hypothesis clause.

Proof.
Proof of (1). For every Horn clause H_i in H, $\Sigma \models H_i$ (by lines 8–13 of Horn-learn, and Lemma 10). Therefore, by Lemma 9, H_i has a target Horn clause.
Proof of (2). First, we show that any new hypothesis clause added to H has a target clause distinct from the target clauses of the other hypothesis clauses in H. Next, we show that if two hypothesis clauses have distinct target clauses at the beginning of an iteration of the loop of lines 4–14, then they still have distinct target clauses at the end of the iteration.

Let H_i be the first hypothesis Horn clause in H for C_j. That is, there is no H_k such that $k < i$ and H_k is a hypothesis Horn clause in H for C_j. Another hypothesis clause $H_{i'}$ with the target clause C_j would have been added to H such that $i' > i$, only if there was a counterexample e belonging to C_j for which H_i is not the correct hypothesis Horn clause (by Lemma 14). That means $C_j \succeq e$ and $C_j \not\succeq lgg(H_i, e)$. This implies, by Lemma 7, $C_j \not\succeq H_i$. That is a contradiction, because H_i is a hypothesis Horn clause for C_j. Therefore, such a $H_{i'}$ cannot exist in H. That is, $H_{i'}$ could have been added only if it had a distinct target clause.

Let C_j and $C_{j'}$ be two distinct target clauses for the clauses H_i and $H_{i'}$ in H, respectively, at the beginning of an iteration of the loop of lines 4–14. That means, $C_j \succeq H_i$ and $C_j \not\succeq H_{i'}$. Also, $C_{j'} \succeq H_{i'}$ and $C_{j'} \not\succeq H_i$.

At most one of H_i and $H_{i'}$ can change in an iteration of the loop. If neither changes, we are done with the proof. Suppose that H_i changes without loss of generality. H_i can change in the lines 8 and 13. We need to show that both these changes maintain that $C_{j'} \not\succeq H_i$. Since $C_{j'} \not\succeq H_i$, $C_{j'} \not\succeq lgg(H_i, e)$ (by Lemma 7). Therefore, line 8 maintains the property. In line 13, Generalize returns a superset of its argument. By Proposition 8, $C_{j'} \not\succeq \mathsf{Generalize}(lgg(H_i, e))$, thus maintaining the property. Therefore, H_i and $H_{i'}$ have different target clauses at the end of the iteration. □

Now to the main theorem on the exact learnability.

Theorem 16. *Function-free non-recursive Horn definitions are exactly learnable using equivalence and membership queries.*

Proof. We prove this theorem by showing that Horn-learn exactly learns function-free non-recursive Horn definitions.

Part 2 of Lemma 15 implies that for every H_i of H, there is a C_j such that $C_j \succeq H_i$. That means, H covers no example that is not covered by the target concept C. In other words, H is never over-general in Horn-learn. Therefore, every counterexample is an example that is covered by Σ, but not by H.

Equivalence query guarantees that whenever Horn-learn gets a new example, it is not already covered by the hypothesis H. At the end of each iteration, before asking an equivalence query, by Lemma 13 and Lemma 14, Horn-learn guarantees that all the previous examples are covered by H. Each example, either modifies an existing hypothesis Horn clause (its correct hypothesis Horn clause) or adds a new Horn clause. The minimum change in H that is required to cover a new example is a change of a variable in its correct hypothesis Horn clause if one exists. That is, each new example, except the ones that add new Horn clauses, contributes at least one variable. Let n be the number of Horn clauses in a concept, l be the maximum number of literals in a clause in the concept, v be the maximum number of variables in a clause in the concept, and k be the number of literals in the largest counterexample. Because Generalize guarantees that each Horn clause in the hypothesis has at most as many literals as there are in its target Horn clause (by Lemma 10 and Lemma 11), the number of variables in each Horn clause is at most v. Part 1 of Lemma 15 guarantees that H has at most n Horn clauses. Therefore, the total number of variables is at most nv. Horn-learn requires n examples to add each of the n Horn clauses in H. It requires at most nv examples to variablize all the Horn clauses in H. Therefore, Horn-learn requires $n(v+1)$ examples, and, hence, $n(v+1)$ equivalence queries.

Let m be the number of hypothesis clauses in the hypothesis H at any time. Then, for each of the base examples that form new Horn clause in H, Horn-learn asks at most m membership queries for deciding that there is no correct hypothesis Horn clause in H, and at most k membership queries to simplify and generalize using Generalize (because there are at most k literals in an example). Each new Horn clause has at most l literals (by Lemma 10). For each of the other examples, at most m membership queries are needed to determine a correct hypothesis Horn clause, and kl (which is the size of lgg) number of membership queries to generalize using Generalize. Therefore, the total number of queries is at most $mn + kn + nv(m + kl)$, which is at most $n^2 + kn + nv(n + kl)$. This is also the upper limit on the running time of the algorithm. □

By the above theorem and the transformation result from the exact learning model to the PAC model [Angluin, 1988], we have the following.

Corollary 17. *Function-free non-recursive Horn definitions are PAC-learnable using membership queries.*

4 Learnability of Goal-Decomposition Rules

In AI planning, domain-specific control knowledge is necessary to make planning task computationally feasible. Goal-decomposition rules (d-rules) is a natural method for representing control knowledge [Reddy, Tadepalli, & Roncagliolo, 1996]. They are similar to hierarchical transition networks [Erol, Hendler, & Nau, 1994].

A d-rule is a 3-tuple $\langle g, c, sg\rangle$ that decomposes a goal g into a sequence of subgoals sg, provided condition c holds in the initial state. Goal g is a positive first-order literal, condition c is a conjunction of positive first-order literals. Subgoals sg are first-order positive literals.

The following is an example of a simple d-rule from Blocks-world (BW) domain:

```
      goal: (on ?x ?y)
  subgoals: ((clear ?x) (clear ?y) (put-on ?x ?z ?y))
conditions: ((block ?x) (block ?y) (table ?z))
```

A goal may have multiple d-rules—i.e., multiple condition-and-subgoals pairs. For example, there could be some other rules for the goal `(on ?x ?y)` with different condition-and-subgoals pairs. We call these disjunctive d-rules. If a goal does not appear anywhere in the conditions or subgoals, then we refer to them as non-recursive d-rules. The above example is a non-recursive d-rule.

In this section, we show that non-recursive disjunctive d-rules are learnable by converting them into a set of function-free non-recursive Horn clauses. The examples for this purpose are positive examples, each of which has a goal, and a sequence of successive states starting from an initial state and leading to a goal state. This can be viewed as specifying the initial condition and a sequence of subgoals, with both including several irrelevant literals. Each component is ground—i.e., it does not contain any variables. The subgoals component, in addition to having irrelevant literals, may have some sequence information missing. In the above example, if block `?x` is already clear in the initial state, then the related example comes with no information on the relative order of the subgoals `(clear ?x)` and `(clear ?y)`.

The conversion of d-rules to Horn-clauses is straightforward, except for the sequences in subgoals components. To represent the sequential order of subgoal literals, we annotate all literals using situation-calculus like notation. Each literal is added the name of the situation it is true in as an additional parameter, and the relative order of situations is specified with new predicates. Since literals for a condition are true in an initial state, they are assigned the first situation. For instance, in our BW example, a training example would be the following.

```
((block a S0) (block b S0) (block c S0) (table Tab S0) (clear a S0)
(clear c S0) (clear a S1) (clear b S1) (put-on a Tab b S2) (notAfter
S0 S0)
(notAfter S0 S1) (notAfter S1 S1) (notAfter S1 S2) (notAfter S2 S2))
→ (on a b S2)
```

From the theorem in the previous section it follows that d-rules along with the relative order of the subgoal literals can be learned from examples and membership queries.

We have implemented an efficient version of this scheme, and demonstrated in [Reddy, Tadepalli, & Roncagliolo, 1996, Reddy & Tadepalli, 1997b] that it can successfully learn d-rules for efficient planning in two other domains: a variation of STRIPS world [Fikes, Hart, & Nilsson, 1972], and Kanfer-Ackerman air-traffic control domain [Ackerman & Kanfer, 1993].

5 Discussion

In this work, we have shown that first-order function-free non-recursive Horn definitions are learnable utilizing reasonable resources and queries. From this, it is a straightforward extension that first-order monotone DNFs are PAC-learnable using membership queries. This is a more general result than the learnability of propositional monotone DNFs [Angluin, 1988]. Another related concept class to Horn definitions is Horn sentences; Horn sentences allow different head literals in a set of Horn clauses, as opposed to single head literal for all Horn clauses in

Horn definitions. It is shown that propositional Horn sentences are exactly learnable from equivalence and membership queries [Angluin, Frazier, & Pitt, 1992, Frazier & Pitt, 1993]. The entailment queries in [Frazier & Pitt, 1993] are similar to our queries, but differ in one crucial aspect. The queries in [Frazier & Pitt, 1993] ask about intermediate predicates in a hierarchy of predicates, whereas our queries concern single-level clauses. Learnability of first-order function-free Horn sentences from entailment and membership queries is an important open question, which we are investigating at present. As mentioned in the introduction, Page [1993] shows that non-recursive and "simple"—that is, only the terms in the head of a clause can appear in the body of the clause— Horn definitions with predicates of fixed arity are PAC-learnable using equivalence and subset queries. This class considered by Page handles functions, whereas the class we considered is function-free.

There is a difference between what an example means in ILP, and what it means in our case along with the cases of [Haussler, 1989] and [Angluin, Frazier, & Pitt, 1992]. An example in ILP comprises a ground head literal and an indication whether it is a positive or a negative example. In our case, it is a conjunction of ground relations (literals) describing a set of objects such that it satisfies a head predicate. Since our motivation is planning in a structural domain, this notion for an example is natural. However, the extended instances in the ILP work in [Cohen, 1995b] are similar to our notion of examples.

The algorithm in Figure 1 is similar in spirit to an ILP system called CLINT [De Raedt & Bruynooghe, 1992] in the sense that they both are incremental and interactive. Like in our algorithm, CLINT uses queries to eliminate irrelevant literals. CLINT raises the generality of hypotheses by proposing more complex hypothesis clauses, whereas our algorithm uses lgg.

Several pieces of research have used the lgg idea in different ways for the purpose of generalization. Haussler [1989] considers learning multiple conjunctive concepts. In the hypothesis language considered by him where the number of objects per scene is fixed, lgg of two hypotheses has at most the size of one of the hypotheses, but is not unique. Haussler uses queries to select an lgg which is in the target. On the other hand, in our case, lgg is unique, but its size grows exponentially in the number of hypotheses of which it is lgg. We use queries to reduce the size of the hypothesis generated by the lgg (see Generalize procedure in Fig. 1). Frazier and Pitt [1996] also use a pruning procedure similar to our Generalize procedure to limit the size of lgg in Classic learning. GOLEM [Muggleton & Feng, 1990] is another system that uses lgg to generalize hypotheses. GOLEM mitigates the problem of combinatorial explosion due to lgg in two ways: (1) by restricting the hypothesis language to ij-determinate Horn clauses which guarantee polynomial-sized lgg; and (2) by using negative examples to elimiate literals from the hypotheses. In the case of [Page, 1993], the simplicity and the fixed arity restrictions make the size of lgg polynomial in the sizes of the hypotheses being generalized.

For concept learning problems, answering membership queries can be unreasonably demanding. However, as mentioned in the introduction, the work in this paper is motivated by the practical application of learning goal-decomposition rules (d-rules) for planning. Note that the membership queries are d-rules hypothesized by the learner. In our application, the d-rule learner can automatically answer these queries by generating random planning problems that must use the hypthesized d-rules, and then trying out these d-rules in solving the planning problems [Reddy & Tadepalli, 1997a, Reddy & Tadepalli, 1997b]. Similar to our application of the learning algorithm to speedup learning in planning, Page [1993] has proposed a way to apply his algorithm to learning LEX's [Mitchell, Utgoff, & Banerji, 1983] control rules.

It is possible that in some cases the membership queries asked by the learner must respect a background domain theory. Consider a case where such queries are answered automatically by a theorem prover or by running a simulator. It is natural to want to restrict the queries so that they are not irrational or inconsistent with the domain theory. One open question is what kinds of restrictions are needed to be placed on the domain theory so that it is possible to learn with only consistent membership queries.

Acknowledgments. We gratefully acknowledge the support of ONR under grant # N00014-95-1-0557 and NSF under grant # IRI-9520243. We thank Roni Khardon and the anonymous reviewers for their insightful comments.

References

[Ackerman & Kanfer, 1993] Ackerman, P., and Kanfer, R. 1993. *Kanfer-Ackerman Air Traffic Control Task© CD-ROM Database, Data-Collection Program, and Playback Program.* Dept. of Psychology, Univ. of Minn., Minneapolis, MN.

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

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

[Cohen, 1995a] Cohen, W. 1995a. Pac-learning non-recursive prolog clauses. *Artificial Intelligence* 79(1):1–38.

[Cohen, 1995b] Cohen, W. 1995b. Pac-learning recursive logic programs: efficient algorithms. *Jl. of AI Research* 2:500–539.

[Cohen, 1995c] Cohen, W. 1995c. Pac-learning recursive logic programs: negative results. *Jl. of AI Research* 2:541–573.

[De Raedt & Bruynooghe, 1992] De Raedt, L., and Bruynooghe, M. 1992. Interactive concept learning and constructive induction by analogy. *Machine Learning* 8(2):107–150.

[Džeroski, Muggleton, & Russell, 1992] Džeroski, S.; Muggleton, S.; and Russell, S. 1992. Pac-learnability of determinate logic programs. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, 128–135.

[Erol, Hendler, & Nau, 1994] Erol, K.; Hendler, J.; and Nau, D. 1994. HTN planning: complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94).* AAAI Press.

[Fikes, Hart, & Nilsson, 1972] Fikes, R.; Hart, P.; and Nilsson, N. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3:251–288.

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

[Frazier & Pitt, 1996] Frazier, M., and Pitt, L. 1996. CLASSIC learning. *Machine Learning* 26:151–194.

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

[Kietz & Lübbe, 1994] Kietz, J.-U., and Lübbe, M. 1994. An efficient subsumption algorithm for inductive logic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, 130–138.

[Kowalski, 1970] Kowalski, R. 1970. The case for using equality axioms in automatic demonstration. In *Lecture Notes in Mathematics*, volume 125. Springer-Verlag.

[Lassez, Maher, & Marriott, 1988] Lassez, J.-L.; Maher, M.; and Marriott, K. 1988. Unification revisited. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann.

[Lloyd, 1987] Lloyd, J. 1987. *Foundations of Logic Programming* (2nd ed.). Berlin: Springer-Verlag.

[Mitchell, Utgoff, & Banerji, 1983] Mitchell, T.; Utgoff, P.; and Banerji, R. 1983. Learning by experimentation: Acquiring and refining problem-solving heuristics. In Michalski, R., and et al., eds., *Machine learning: An artificial intelligence approach*, volume 1. Morgan Kaufmann.

[Muggleton & Feng, 1990] Muggleton, S., and Feng, C. 1990. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, 368–381. Ohmsha/Springer-Verlag.

[Nienhuys-Cheng & de Wolf, 1996] Nienhuys-Cheng, S.-H., and de Wolf, R. 1996. Least generalizations and greatest specializations of sets of clauses. *Jl. of AI Research* 4:341–363.

[Page, 1993] Page, C. 1993. *Anti-Unification in Constraint Logics: Foundations and Applications to Learnability in First-Order Logic, to Speed-up Learning, and to Deduction*. Ph.D. Dissertation, University of Illinois, Urbana, IL.

[Plotkin, 1970] Plotkin, G. 1970. A note on inductive generalization. In Meltzer, B., and Michie, D., eds., *Machine Intelligence*, volume 5. New York: Elsevier North-Holland. 153–163.

[Reddy & Tadepalli, 1997a] Reddy, C., and Tadepalli, P. 1997a. Inductive logic programming for speedup learning. To appear in IJCAI-97 workshop on Frontiers of ILP.

[Reddy & Tadepalli, 1997b] Reddy, C., and Tadepalli, P. 1997b. Learning goal-decomposition rules using exercises. In *Proceedings of the 14th International Conference on Machine Learning*. Morgan Kaufmann.

[Reddy, Tadepalli, & Roncagliolo, 1996] Reddy, C.; Tadepalli, P.; and Roncagliolo, S. 1996. Theory-guided empirical speedup learning of goal-decomposition rules. In *Proceedings of the 13th International Conference on Machine Learning*, 409–417. Morgan Kaufmann.

[Schapire, 1990] Schapire, R. 1990. The strength of weak learnability. *Machine Learning* 5:197–227.

This article was processed using the LaTeX macro package with LLNCS style