

# The Subsumption Lattice and Query Learning<sup>\*</sup>

Marta Arias and Roni Khardon

Department of Computer Science, Tufts University  
Medford, MA 02155, USA  
{marias,roni}@cs.tufts.edu

**Abstract.** The paper identifies several new properties of the lattice induced by the subsumption relation over first-order clauses and derives implications of these for learnability. In particular, it is shown that the length of subsumption chains of function free clauses with bounded size can be exponential in the size. This suggests that simple algorithmic approaches that rely on repeating minimal subsumption-based refinements may require a long time to converge. It is also shown that with bounded size clauses the subsumption lattice has a large branching factor. This is used to show that the class of first-order length-bounded monotone clauses is not properly learnable from membership queries alone. Finally, the paper studies pairing, a generalization operation that takes two clauses and returns a number of possible generalizations. It is shown that there are clauses with an exponential number of pairing results which are not related to each other by subsumption. This is used to show that recent pairing-based algorithms can make exponentially many queries on some learning problems.

## 1 Introduction

The field of Inductive Logic Programming (ILP) is concerned with developing theory and methods that allow for efficient learning of classes of concepts expressed in the language of first-order logic. Subsumption is a generality relation over first order clauses that induces a quasi-order on the set of clauses. The subsumption lattice is of crucial importance since many ILP algorithms perform a search over this space and as a result the lattice has been investigated extensively in the literature (see survey in [15]). The paper contributes to this study in two ways. First, we expose and prove new properties of the subsumption lattice of first-order clauses. Second, we use these properties to prove negative learning results in the model of exact learning from queries. These results illustrate the connection between the subsumption lattice and learning.

This work arises from the study of query complexity of learning in first order logic. Several positive learnability results exist in the model of exact learning from queries [1]. However, except for a “monotone-like case” [19] the query complexity is either exponential in one of the crucial parameters (e.g. the number of

---

<sup>\*</sup> This work has been partly supported by NSF Grant IIS-0099446

universally quantified variables) [13, 3] or the algorithms use additional syntax-based oracles [7, 20, 18]. It is not clear whether the exponential dependence is necessary or not. Previous work in [4] showed that the VC-dimension cannot resolve this question. The current paper explores how properties of subsumption affect this question.

We start by considering the length of proper subsumption chains  $c_1 \prec c_2 \prec \dots \prec c_n$  of first order clauses of restricted size. This is motivated by two issues. First, many ILP algorithms (e.g. [21, 17, 8]) use refinement of clauses where in each step the clause is modified using a minimal subsumption step. Thus the length of subsumption chains hinges on convergence of such approaches. A second motivation comes from the use of certificates [12, 11] to study query complexity. It is known [12, 11] that a class  $\mathcal{C}$  is learnable from equivalence and membership queries if and only if the class  $\mathcal{C}$  has polynomial certificates. Previous work in [6] developed certificates for propositional classes. In particular, one of the constructions of certificates for Horn expressions uses the fact that all proper subsumption chains of propositional Horn clauses are short. Hence any generalization of this construction to first order logic relies on the length of such chains.

Section 3 shows that subsumption chains can be exponentially long (in number of literals and variables) even with function free clauses with a bounded number of literals. This result suggests that simple algorithmic approaches that rely only on minimal refinement steps may require a long time to converge and excludes simple generalizations of the certificate construction. We also show that if one imposes inequalities on all terms in a clause then subsumption chains are short. This further supports the use and study of inequated expressions as done e.g. in [13, 3, 9].

The chain length result gives an informal argument against certain approaches. Section 4 uses a similar construction to show that the class of length-bounded monotone first order clauses is not properly learnable using membership queries only. This result is derived by studying the lattice structure of length bounded clauses and using it to show that the *teaching dimension* [2, 10] is exponential in the size. The result follows since the teaching dimension gives a lower bound for the number of membership queries required to learn a class [2, 10].

Finally in Section 5 we address the complexity of the algorithms given in [13, 3] discussed above. One of the sources of exponential dependence on the number of variables is the number of pairings. Intuitively, a pairing is an operation that, given two first-order clauses, results in a new clause which is more general than the initial ones; two clauses have many pairings and the algorithm enumerates these in the process of learning. Results in [13, 3] gave an upper-bound on the number of pairings, but left it open whether a large number of pairings can actually occur in examples. We give an exponential lower bound (in number of variables) on the number of pairings and construct an explicit example showing that the algorithm can be forced to make an exponential number of queries.

Due to space limitations several proofs are omitted from the paper; they can be found in [5].

## 2 Preliminaries

We assume familiarity with basic concepts in first order logic as described e.g. in [14, 15]. We briefly review notions relevant to this paper.

A signature  $\mathcal{S}$  consists of a finite set of predicates  $P$  and a finite set of functions  $F$ , both with their associated arities. Constants are functions with arity 0. A countable set of variables  $x_1, x_2, x_3, \dots$  is used to construct expressions. A variable is a *term*. If  $t_1, \dots, t_n$  are terms and  $f \in F$  is a function symbol of arity  $n$ , then  $f(t_1, \dots, t_n)$  is a term. An *atom* is an expression  $p(t_1, \dots, t_n)$  where  $p \in P$  is a predicate symbol of arity  $n$  and  $t_1, \dots, t_n$  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 universally quantified. A *Horn clause* has at most one positive literal and an arbitrary number of negative literals. A Horn clause  $\neg p_1 \vee \dots \vee \neg p_n \vee p_{n+1}$  is equivalent to its implicational form  $p_1 \wedge \dots \wedge p_n \rightarrow p_{n+1}$ . We call  $p_1 \wedge \dots \wedge p_n$  the *antecedent* and  $p_{n+1}$  the *consequent* of the clause. A *meta-clause* is a pair of the form  $[s, c]$ , where both  $s$  and  $c$  are sets of atoms such that  $s \cap c = \emptyset$ ;  $s$  is the *antecedent* of the meta-clause and  $c$  is the *consequent*. Both are interpreted as the conjunction of the atoms they contain. Therefore, the meta-clause  $[s, c]$  is interpreted as the logical expression  $\bigwedge_{b \in c} s \rightarrow b$ . An ordinary clause  $C = s_c \rightarrow b_c$  corresponds to the meta-clause  $[s_c, \{b_c\}]$ . Fully inequated clauses [3] are clauses whose terms are forced to be always distinct. That is, any instance of a fully inequated clause is not allowed to unify any of its terms. This can be done by adding explicit inequalities on all terms as in:  $E = [x \neq f(x)] \wedge [x \neq a] \wedge [a \neq f(x)] \wedge p(x, f(x)) \wedge p(a, x) \rightarrow q(a)$ .

We use the symbol ‘ $\models$ ’ to denote logical implication which is defined following the standard semantics of first-order logic.

We need several parameters to quantify the complexity of a first-order expressions; we use the first-order expression  $E = \neg p(x, f(x)) \vee \neg p(a, b) \vee q(b)$  to illustrate these.  $NTerms(\cdot)$ : counts the number of distinct terms in the input expression. Hence,  $NTerms(E) = 4$  corresponding to the term set  $\{x, a, f(x), b\}$ .  $WTerms(\cdot)$ : similar to  $NTerms$ , with the only difference that functional terms are given twice as much weight as variables. Hence,  $WTerms(E) = 7$  since terms in  $\{a, f(x), b\}$  contribute 2 and  $x$  contributes 1.  $NLiterals(\cdot)$ : counts the number of literals in the input expression. Hence,  $NLiterals(E) = 3$ .

Let  $C, D$  be two arbitrary first-order clauses. We say that a clause  $C$  *subsumes* a clause  $D$  and denote this by  $C \preceq D$  if there is a substitution  $\theta$  such that  $C \cdot \theta \subseteq D$ . Moreover, they are *subsume-equivalent*, denoted  $C \sim D$ , if  $C \preceq D$  and  $D \preceq C$ . We say that  $C$  *strictly or properly subsumes*  $D$ , denoted  $C \prec D$ , if  $C \preceq D$  but  $D \not\preceq C$ . The relation  $\preceq$  is reflexive and transitive and hence it induces a quasi-order on the set of clauses.

## 3 On the Length of Proper Chains

In this section we study the length of proper subsumption chains of clauses  $c_1 \prec c_2 \prec \dots \prec c_n$ . It is known that infinite chains exist if one does not restrict

clause size [16, 15] but bounds for clauses of restricted size (which are necessarily finite) were not known before. We show that in the case of fully inequated clauses, the length of any proper chain is polynomial in the number of literals and the number of terms in the clauses involved. On the other hand, if clauses are not fully inequated, then chains of length exponential in the number of variables (or literals) exist, even if clauses are function free.

### 3.1 Subsumption Chains for Fully Inequated Clauses are Short

We say that a substitution  $\theta$  is unifying w.r.t. a clause  $c_1$  if there exist two distinct terms  $t, t'$  in  $c_1$  that have been syntactically unified i.e.  $t \cdot \theta = t' \cdot \theta$ . It is easy to verify the following lemma:

**Lemma 1.** *Let  $c_1, c_2$  be two fully inequated clauses. If  $c_1 \preceq c_2$ , then (1) it must be via a non-unifying substitution w.r.t.  $c_1$ , (2)  $WTerms(c_1) \leq WTerms(c_2)$ , and (3)  $NLiterals(c_1) \leq NLiterals(c_2)$ .*

**Lemma 2.** *Let  $c_1, c_2$  be fully inequated clauses such that  $c_1 \prec c_2$ . Then, either  $NLiterals(c_1) < NLiterals(c_2)$  or  $WTerms(c_1) < WTerms(c_2)$ .*

*Proof.* By the previous lemma we only need to disprove the possibility that both  $NLiterals(c_1) = NLiterals(c_2)$  and  $WTerms(c_1) = WTerms(c_2)$ . Suppose so, and let  $\theta$  be the substitution such that  $c_1\theta \subseteq c_2$ . Then  $\theta$  induces a 1-1 mapping of terms. Now if  $\theta$  maps a variable to a non-variable term then  $WTerms(c_1) < WTerms(c_2)$ . So  $\theta$  must be a variable renaming. If  $\theta$  is a variable renaming and  $NLiterals(c_1) = NLiterals(c_2)$ , then  $c_1$  and  $c_2$  must be syntactic variants, contradicting the assumption that  $c_2 \not\preceq c_1$ .  $\square$

As a result each step in a strict subsumption chain reduces one of  $NLiterals$  or  $WTerms$  and since these are bounded by  $2t$  and  $l$  respectively we get:

**Theorem 1.** *The longest proper subsumption chain of fully inequated clauses with at most  $t$  terms and  $l$  literals is of length at most  $2t + l$ .*

### 3.2 Function Free Clauses Have Long Proper Chains

In this section we demonstrate that function free first-order clauses can produce chains of exponential length. We start with a simple construction where the arity of predicates is not constant.

Let  $p$  be a predicate symbol of arity  $a$ . The chain  $d_1 \succ d_2 \succ \dots \succ d_n$  is defined inductively. The first clause is  $d_1 = p(z, \dots, z)$ , and given a clause  $d_i = p_1, p_2, \dots, p_k$ , we define the next clause  $d_{i+1}$  as follows: (1) if  $p_1$  contains only two occurrences of the variable  $z$ , then  $d_{i+1} = p_2, \dots, p_k$ , or else (2) if  $p_1$  contains  $c \geq 3$  occurrences of the variable  $z$ , replace the atom  $p_1$  by a new set of atoms  $p'_1, \dots, p'_{k'}$  such that  $k' = \min(c, l - k + 1)$ , and every new atom  $p'_j$  for  $1 \leq j \leq k'$  is a copy of  $p_1$  in which the  $j$ 'th occurrence of the variable  $z$  has been replaced by a new fresh variable not appearing in  $d_i$  (the same variable for all copies).

*Example 1.* Suppose  $p$  has arity 4 and that  $l = 3$ . The construction produces the following chain of length 11:

$$\begin{array}{l}
p(z, z, z, z) \\
\succ p(x_1, z, z, z), p(z, x_1, z, z), p(z, z, x_1, z) \\
\succ p(x_1, x_2, z, z), p(z, x_1, z, z), p(z, z, x_1, z) \\
\succ p(z, x_1, z, z), p(z, z, x_1, z) \\
\succ p(x_2, x_1, z, z), p(z, x_1, x_2, z), p(z, z, x_1, z) \\
\succ p(z, x_1, x_2, z), p(z, z, x_1, z) \\
\succ p(z, z, x_1, z) \\
\succ p(x_2, z, x_1, z), p(z, x_2, x_1, z), p(z, z, x_1, x_2) \\
\succ p(z, x_2, x_1, z), p(z, z, x_1, x_2) \\
\succ p(z, z, x_1, x_2) \\
\succ \emptyset
\end{array}$$

Let  $N(c, s)$  be the number of subsumption generalizations that can be produced by this method when starting with a singleton clause which is allowed to expand on  $s$  literals (i.e.,  $l = s + 1$ ) and whose only atom has  $c \geq 2$  occurrences of the variable  $z$ . Then, the following relations hold: (A)  $N(2, s) = 1$ , for all  $s \geq 0$ . To see this note that when there are only 2 occurrences of the variable  $z$ , the only possible step is to remove the atom, thus obtaining the empty clause. (B)  $N(c, 0) = c - 1$ , for all  $c \geq 2$ . This is derived by observing that when we have  $c \geq 2$  occurrences of the distinguished variable  $z$  and no expansion on the number of literals is possible, we can apply  $c - 2$  steps that replace occurrences of  $z$  by new variables, and a final step that drops the literal. After this, no more generalizations are possible. (C)  $N(c, s) = 1 + \sum_{i=\max(0, s-c+1)}^s N(c-1, i)$ , for all  $c > 2, s > 0$ . This recurrence is obtained by observing that the initial clause containing our single atom can be replaced by  $\min(c, s + 1)$  ‘‘copies’’ in a first generalization step leaving  $q = \max(0, s - c + 1)$  empty slots. After this, each of these copies which contain  $c - 1$  occurrences of the distinguished variable  $z$ , go through the series of generalizations: the left-most atom has  $q$  positions to use for its expansion and is generalized  $N(c - 1, q)$  times until it is finally dropped; the next atom has  $q + 1$  position to expand since the left-most atom has been dropped, and hence it produces  $N(c - 1, q + 1)$  generalization steps until it is finally dropped, and so on. The next lemma can be proved by induction on  $c$  and  $s$ .

**Lemma 3.**  $N(c, s) \geq \binom{c}{s+1} - 1$  for  $c \geq 2$  and  $s \geq 0$ .

It remains to show that this is a proper chain. First, we investigate key structural properties of the clauses participating in our chain. The following three lemmas can be proved by induction on the updates of  $d_i$ .

**Lemma 4.** Let  $\text{Vars}(p)$  be the variables occurring in the atom  $p$ . For all  $d_i = p_1, \dots, p_k$  the following properties hold: (1) Every atom  $p_j \in d_i$  contains no repeated occurrences of variables, with the exception of  $z$ , which appears at least twice in each atom. (2)  $\text{Vars}(p_j) \supseteq \text{Vars}(p_{j+1})$  for all  $j = 1, \dots, k - 1$ .

From the properties stated in the previous lemma, it follows that we can view any clause  $d_i$  as a sequence of blocks of atoms  $B_1, B_2, \dots, B_m$  such that all the atoms in a single block contain exactly the same variables, and variables appearing in neighboring blocks are such that  $\text{Vars}(B_j) \supset \text{Vars}(B_{j+1})$ .

**Lemma 5.** *Fix some clause  $d_i$ , and let  $p$  be an atom in any block  $B$ . If  $p \cdot \theta \in B$ , then  $\theta$  does not change variables in  $p$ .*

**Lemma 6.** *Let  $d_i$  be any clause in the sequence and let  $B_1, \dots, B_m$  be its blocks. Then, for any pair of blocks  $B_{i_1}$  and  $B_{i_2}$  s.t.  $i_1 < i_2$ , there exists some variable in  $\text{Vars}(B_{i_2}) \setminus \{z\}$  that is in the same position  $j$  in all the atoms in  $B_{i_1}$  but in all the atoms in  $B_{i_2}$  it appears in different positions, always different from the one in  $B_{i_1}$ . Moreover, all the atoms in  $B_{i_2}$  contain the variable  $z$  at position  $j$ .*

**Lemma 7.** *Fix some clause  $d_i = p_1, \dots, p_k$  with at least 2 atoms (i.e.,  $k \geq 2$ ). Then  $(p_2, \dots, p_k) \cdot \theta \subseteq d_i$  only if  $\theta$  does not change variables in  $p_2$ .*

*Proof.* Let  $d_i = B_1, \dots, B_m$ . Let  $p$  be any atom in any block  $B_j$  and assume  $p \cdot \theta \in d_i$ . Notice that  $p \cdot \theta \notin B_1, \dots, B_{j-1}$  since atoms in blocks  $B_1, \dots, B_{j-1}$  contain strictly more variables than  $p \cdot \theta$ . Hence  $p \cdot \theta \in B_j, \dots, B_m$ . We first claim that if  $\theta$  does not change variables in  $B_{j+1}, \dots, B_m$  then  $\theta$  does not change variables in  $B_j$ . To prove the claim note that if  $p \cdot \theta \in B_j$  then Lemma 5 applies. On the other hand if  $p \cdot \theta \in B_{j'}$  for  $j < j'$  then Lemma 6 guarantees that there exists some variable  $x \in \text{Vars}(B_{j'})$  that appears in a position in  $p$  in which atoms in  $B_{j'}$  contain the variable  $z$ . Since by assumption  $\theta$  does not change this variable this implies that  $p \cdot \theta \notin B_{j'}$  leading to a contradiction.

Finally if  $(p_2, \dots, p_k) \cdot \theta \subseteq d_i$  then we have an atom  $p$  as above from each block. Therefore we can apply the claim inductively starting with  $j = m$  and until  $j = i_2$  where  $i_2$  is the block index of  $p_2$ . This implies that  $\theta$  does not change variables that appear in the leftmost block of  $p_2, \dots, p_k$ , and hence in  $p_2$  as required.  $\square$

**Lemma 8.** *For all  $i = 1, \dots, n - 1$  we have that  $d_i \succ d_{i+1}$ .*

*Proof.* Suppose that  $d_i = p_1, \dots, p_k$ . We have the following possible transitions from  $d_i$  to  $d_{i+1}$ :

Case 1.  $d_{i+1} = p_2, \dots, p_k$ . Clearly,  $d_i \supset d_{i+1}$ , and hence  $d_i \succeq d_{i+1}$  via the empty substitution. Suppose by way of contradiction that  $d_i \preceq d_{i+1}$ , so there must be a substitution  $\theta$  s.t.  $d_i \cdot \theta \subseteq d_{i+1}$ . Clearly,  $i + 1 \neq n$  since otherwise we could not satisfy  $d_i \cdot \theta \subseteq d_{i+1} = \emptyset$ . Therefore,  $d_{i+1} \neq \emptyset$  and  $d_i$  contains at least 2 atoms. The fact  $d_i \cdot \theta \subseteq d_{i+1}$  implies that  $(p_2, \dots, p_k) \cdot \theta \subseteq d_i$ , and by Lemma 7,  $\theta$  must not change variables in  $p_2$ . If  $p_1$  and  $p_2$  are in the same block, then  $p_1 \cdot \theta = p_1 \notin d_{i+1}$ . If  $p_1$  and  $p_2$  are in different blocks, then Lemma 6 guarantees that for every atom in  $p_2, \dots, p_k$  there is a variable that appears in a different location in  $p_1$  and as above this variable cannot be changed by  $\theta$ . Hence,  $p_1 \cdot \theta \notin d_{i+1}$ , contradicting our assumption that  $d_i \preceq d_{i+1}$ .

Case 2.  $d_{i+1} = p'_1, \dots, p'_{k'}, p_2, \dots, p_k$ . Let  $x$  be the newly introduced variable. Then,  $d_{i+1} \cdot \{x \mapsto z\} \subseteq d_i$  and hence  $d_i \succeq d_{i+1}$ . To see that  $d_i \not\preceq d_{i+1}$ , suppose

that this is not the case. Hence, there must be a substitution  $\theta$  such that  $d_i \cdot \theta \subseteq d_{i+1}$ . If  $d_i = p_1$ , (i.e.,  $d_i$  contains one atom only), then  $p_1 \cdot \theta \subseteq p'_1, \dots, p'_{k'}$ . In this case,  $\theta$  must map  $z$  into the new variable  $x$  but this results in multiple occurrences of  $x$ , and hence  $p_1 \cdot \theta \not\subseteq p'_1, \dots, p'_{k'}$ . Hence,  $d_i$  must contain at least two atoms and the substitution  $\theta$  must satisfy that  $(p_1, \dots, p_k) \cdot \theta \subseteq p'_1, \dots, p'_{k'}, p_2, \dots, p_k$ . The new atoms  $p'_1, \dots, p'_{k'}$  contain more variables than  $p_1, \dots, p_k$ , therefore  $(p_1, \dots, p_k) \cdot \theta \subseteq p_2, \dots, p_k$ . By the same reasoning as in the previous case, we conclude that  $d_i \succ d_{i+1}$ .  $\square$

Now Lemma 3 and Lemma 8 imply:

**Theorem 2.** *Let  $p$  be a predicate symbol of arity  $\alpha \geq 1$ . There exists a proper subsumption chain of length  $\binom{\alpha}{1}$  of function free clauses using at most  $\alpha$  variables and  $l$  literals.*

The construction above can be improved to use predicates of arity 3 as follows.

**Definition 1.** *Let  $d$  be any clause. Let  $Trans(d)$  be the clause obtained by replacing each literal  $p(t_1, \dots, t_a)$  with a new set  $\{p(y_i, y_{i+1}, t_i) \mid 1 \leq i \leq a\}$ , where all  $y_1, \dots, y_{a+1}$  are new variables not appearing in  $d$ . The new variables  $y_1, \dots, y_{a+1}$  are different for each atom in  $d$ .*

*Example 2.* The clause  $p(z, x_1, x_2, z), p(z, z, x_1, z)$  is transformed into the clause  $p(y_1, y_2, z), p(y_2, y_3, x_1), p(y_3, y_4, x_2), p(y_4, y_5, z), p(y'_1, y'_2, z), p(y'_2, y'_3, z), p(y'_3, y'_4, x_1), p(y'_4, y'_5, z)$ .

Consider a function free clause  $d$  with predicate symbols of arity at most  $a$ , containing  $v$  variables and  $l$  literals. Then,  $Trans(d)$  uses predicates of arity 3, has  $l(a+1)+v$  variables and  $al$  literals. The next lemma gives the main property of this transformation:

**Lemma 9.** *Let  $d_1, d_2$  be clauses. Then,  $d_1 \preceq d_2$  iff  $Trans(d_1) \preceq Trans(d_2)$ .*

*Proof.* Assume first that  $d_1 \preceq d_2$ , i.e., there is a substitution  $\theta$  from variables in  $d_1$  into terms of  $d_2$  such that  $d_1 \cdot \theta \subseteq d_2$ . Obviously,  $\theta$  does not alter the value of the new variables added to  $Trans(d_1)$ , and hence  $Trans(d_1) \cdot \theta = Trans(d_1 \cdot \theta) \subseteq Trans(d_2)$ , so that  $Trans(d_1) \preceq Trans(d_2)$ .

For the other direction, assume that there exists a substitution  $\theta$  such that  $Trans(d_1) \cdot \theta \subseteq Trans(d_2)$ . Let  $d_1 = l_1^1 \vee l_1^2 \vee \dots \vee l_1^{k_1}$  and let  $\{y_1^j, \dots, y_{arity(l_1^j)+1}^j\}$  be the variables used in the transformation for literal  $l_1^j$  in  $d_1$ , for  $1 \leq j \leq k_1$ . Similarly, let  $d_2 = l_2^1 \vee l_2^2 \vee \dots \vee l_2^{k_2}$  and let  $\{y_1^j, \dots, y_{arity(l_2^j)+1}^j\}$  be the variables used in the transformation for literal  $l_2^j$  in  $d_2$ , for  $1 \leq j \leq k_2$ . First we show that  $\theta$  must map blocks of auxiliary variables in  $Trans(d_1)$ ,  $\{y_1^j, \dots, y_{arity(l_1^j)+1}^j\}$  into blocks of auxiliary variables in  $Trans(d_2)$ ,  $\{y_1^{j'}, \dots, y_{arity(l_2^{j'})+1}^{j'}\}$  so that the predicate symbol of  $l_1^j$  coincides with the predicate symbol of  $l_2^{j'}$ . Moreover, the order of the variables is preserved, i.e.,  $\theta$  maps each  $y_i^j \mapsto y_i^{j'}$ , for all  $1 \leq$

$i \leq \text{arity}(l_2^j)$ . By way of contradiction, suppose that there exists a pair of variables in  $\text{Trans}(d_1)$ ,  $y_i^j$  and  $y_{i+1}^j$ , that have been mapped into  $y_*^a$  and  $y_*^b$ , respectively, where  $a \neq b$ . Then,  $p(y_i^j, y_{i+1}^j, *) \cdot \theta = p(y_*^a, y_*^b, *) \in \text{Trans}(d_2)$ . This contradicts the fact that, by construction, all literals in  $\text{Trans}(d_2)$  are such that the superscripts of the first two auxiliary variables coincide.

Suppose now that some  $y_i^j$  has been mapped into  $y_{i'}^{j'}$  where  $i \neq i'$  and  $i$  is the smallest such index. Assume also that the predicate symbol corresponding to literal  $l_1^j$  is  $p$ . If  $i > 1$ , then  $p(y_{i-1}^j, y_i^j, *) \cdot \theta = p(y_{i-1}^{j'}, y_{i'}^{j'}, *) \in \text{Trans}(d_2)$ . But this is a contradiction since all literals in  $\text{Trans}(d_2)$  are such that its two initial arguments have the form  $p(y_h^*, y_{h+1}^*, *)$  and here  $(i-1)+1 \neq i'$ . If  $i = 1$ , then since  $i' > 1$  there must be an index  $h$  s.t.  $\theta$  maps  $y_{i+h}^j \mapsto y_{i'+h}^{j'}$  but  $\theta$  does not map  $y_{i+h+1}^j \mapsto y_{i'+h+1}^{j'}$ . Thus we arrive to the same contradiction as in the previous case.

Now, the fact that each  $y_i^j \mapsto y_{i'}^{j'}$  implies that  $\theta$  maps arguments of literals in  $d_1$  into arguments in the same position of literals in  $d_2$ . Moreover, since blocks of variables are not mixed, all arguments from a literal in  $d_1$  are mapped into all the arguments of a fixed literal in  $d_2$ , so we conclude that  $d_1 \cdot \theta \subseteq d_2$ .  $\square$

**Theorem 3.** *If there is a predicate symbol of arity at least 3, then there exist proper subsumption chains of length at least  $2^{\sqrt{v}/2}$  of function free clauses using at most  $v$  variables and  $\frac{v}{2}$  literals, where  $v \geq 9$ .*

*Proof.* Theorem 2 shows that there exists a chain of length  $\binom{a}{l} = \binom{\sqrt{v}}{\sqrt{v}/2} > 2^{\sqrt{v}/2}$  if we use predicate symbols of arity  $\sqrt{v}$ ,  $\sqrt{v}$  variables and  $\frac{\sqrt{v}}{2}$  atoms per clause. Consider the chain  $\text{Trans}(d_1) \succ \text{Trans}(d_2) \succ \dots \succ \text{Trans}(d_n)$ . Lemma 9 guarantees that this is also a proper chain. The chain has clauses with  $\frac{\sqrt{v}}{2}(\sqrt{v} + 1) + \sqrt{v} = \frac{v}{2} + \frac{3\sqrt{v}}{2} \leq v$  variables (here we use  $v \geq 9$ ) and  $\sqrt{v} \frac{\sqrt{v}}{2} = \frac{v}{2}$  literals.  $\square$

## 4 Learning from Membership Queries Only

The previous result suggests that simple use of minimal refinement steps may require long time to converge. We next use a related construction to show that there can be no polynomial algorithm that properly learns the class of monotone function-free and length-bounded clauses from membership queries only. We use a combinatorial notion, the *teaching dimension* [2, 10], that is known to be a lower bound for the complexity of exact learning from membership queries only.

**Definition 2.** *The teaching dimension of a class  $\mathcal{T}$  is the minimum integer  $d$  such that for each expression  $f \in \mathcal{T}$  there is a set  $T$  of at most  $d$  examples (the teaching set) with the property that any expression  $g \in \mathcal{T}$  different from  $f$  is not consistent with  $f$  over the examples in  $T$ .*

Let  $k$  be such that  $\log_2 k$  is an integer. Then  $\langle t_1, \dots, t_k \rangle$  denotes the term represented by a complete binary tree of applications of a binary function symbol  $f$

of depth  $\log k$  with leaves  $t_1, \dots, t_k$ . For example,  $\langle 1, 2, 3, 4, 5, 6, 7, 8 \rangle$  represents the term  $f(f(f(1, 2), f(3, 4)), f(f(5, 6), f(7, 8)))$ . Notice that the number of distinct terms in  $\langle t_1, \dots, t_k \rangle$  is at most  $k + \sum_{i=1}^k NTerms(t_i)$ . In particular, if each  $t_i$  is either a variable or a constant, then  $NTerms(\langle t_1, \dots, t_k \rangle) \leq 2k$ .

Let  $p$  be a unary predicate symbol. Consider the clause  $p(\langle a, \dots, a \rangle)$ , where the constant  $a$  occurs  $k$  times. We consider all the possible minimal generalizations of  $p(\langle a, \dots, a \rangle)$ . That is, clauses  $C$  that are strict generalizations of  $p(\langle a, \dots, a \rangle)$  for which no other clause  $C'$  is such that  $p(\langle a, \dots, a \rangle) \succ C' \succ C$ . Among them we find the clauses

$$\begin{aligned}
C_k &= p(\langle x, \dots, x \rangle) \\
C_{k-1} &= p(\langle a, x, \dots, x \rangle) \vee p(\langle x, a, x, \dots, x \rangle) \vee \dots \vee p(\langle x, \dots, x, a \rangle) \\
C_{k-2} &= p(\langle a, a, x, \dots, x \rangle) \vee p(\langle a, x, a, x, \dots, x \rangle) \vee \dots \vee p(\langle x, \dots, x, a, a \rangle) \\
&\vdots \\
C_{k/2} &= p(\langle a, \dots, a, x, \dots, x \rangle) \vee \dots \vee p(\langle x, \dots, x, a, \dots, a \rangle) \\
&\vdots \\
C_1 &= p(\langle a, \dots, a, x \rangle) \vee p(\langle a, \dots, a, x, a \rangle) \vee \dots \vee p(\langle x, a, \dots, a \rangle)
\end{aligned}$$

where each  $C_i$  includes all possibilities of replacing  $i$  positions with a variable. Clearly,  $|C_i| = \binom{k}{i}$ . In particular,  $|C_{k/2}| = \binom{k}{k/2} > 2^{k/2}$ .

We next define the learning problem for which we find an exponential lower bound. The signature  $\mathcal{S}$  consists of the function symbol  $f$  of arity 2, two constants  $a, b$ , and a single predicate symbol  $p$  of arity 1. Fix  $l$  to be some integer. Let the (representation) concept class be  $\mathcal{C} = \{\text{first-order monotone } \mathcal{S}\text{-clauses with at most } l \text{ atoms}\}$  and the set of examples be  $\mathcal{E} = \{\text{first order ground monotone } \mathcal{S}\text{-clauses with at most } l \text{ atoms}\}$ .

We identify the representation concept class  $\mathcal{C}$  with its denotations in the following way. The concept represented by  $C \in \mathcal{C}$  is  $\{E \in \mathcal{E} \mid C \models E\}$  which in this case coincides with  $\{E \in \mathcal{E} \mid C \preceq E\}$ . Thus, this problem is cast in the framework of learning from entailment.

Suppose that the target concept is  $f = p(\langle a, \dots, a \rangle)$  and that  $l \leq \frac{\binom{k}{k/2}}{2}$ . We want to find a minimal teaching set  $T$  for  $f$ . The cardinality of a minimal teaching set for  $f$  is clearly a lower bound on the teaching dimension of  $\mathcal{C}$ . By definition, the examples in  $T$  have to eliminate every other expression in  $\mathcal{C}$ . In other words, for every expression  $g$  in  $\mathcal{C}$  other than  $f$ ,  $T$  must include an example  $E$  such that  $f \preceq E$  and  $g \not\preceq E$  or vice versa.

We first observe that the clause  $C_{k/2}$  is not included in our concept class  $\mathcal{C}$  because it contains too many literals:  $l \leq \frac{\binom{k}{k/2}}{2} = \frac{|C_{k/2}|}{2} < |C_{k/2}|$ . However, subsets of  $C_{k/2}$  with exactly  $l$  atoms are included in  $\mathcal{C}$  because they are monotone  $\mathcal{S}$ -clauses of at most  $l$  literals. Note also that each clause includes  $\leq 2kl$  terms. There are  $K = \binom{\binom{k}{k/2}}{l} > \left(\frac{2^{k/2}}{l}\right)^l = 2^{\Omega(lk)}$  such subsets where we use an additional

restriction that  $l \leq k$ . Let these be  $C_{k/2}^1, \dots, C_{k/2}^K$ . By definition, the teaching set  $T$  has to reject each one of these  $K$  clauses.

Notice that  $C_{k/2}^j \preceq f = p(\langle a, \dots, a \rangle)$  for each  $j = 1, \dots, K$  (consider the witnessing substitution  $\{x \mapsto a\}$ ). Now, to reject an arbitrary  $C_{k/2}^j$ ,  $T$  has to include some example  $E \in \mathcal{E}$  s.t.  $C_{k/2}^j \preceq E$  but  $p(\langle a, \dots, a \rangle) \not\preceq E$ . The only example in  $\mathcal{E}$  that qualifies is  $E^j = C_{k/2}^j \cdot \{x \mapsto b\}$ . Hence, for each  $C_{k/2}^j$  the example  $E^j$  must be included in  $T$  and these examples are distinct. Hence,  $T$  must contain all the examples in  $E^1, \dots, E^K$ . Substituting  $k = \sqrt{t}$  and  $l \leq \frac{\sqrt{t}}{2}$  so that  $2kl \leq t$  we obtain:

**Theorem 4.** *Let  $\mathcal{C}$  be the class of monotone clauses built from a signature containing 2 constants, a binary function symbol and a unary predicate symbol with at most  $l \leq \frac{\sqrt{t}}{2}$  literals and  $t$  terms per clause. Then, the teaching dimension of  $\mathcal{C}$  is  $2^{\Omega(l\sqrt{t})}$ .*

## 5 On the Number of Pairings

Plotkin [16] (see also [15]) defined the *least general generalization* (*lgg*) of clauses w.r.t. subsumption and gave an algorithm to compute it. The *lgg* of  $C_1, C_2$  is a clause  $C$  that subsumes both clauses, namely  $C \preceq C_1, C \preceq C_2$ , and is the least such clause, that is  $D \preceq C$  for any  $D$  that subsumes both clauses. The algorithm essentially takes a cross product of atoms with the same predicate symbol in the two clauses and generalizes arguments bottom up. Generalization of arguments is defined as follows. The *lgg* of two terms  $f(s_1, \dots, s_n)$  and  $g(t_1, \dots, t_m)$  is the term  $f(\text{lgg}(s_1, t_1), \dots, \text{lgg}(s_n, t_n))$  if  $f = g$  and  $n = m$ . Otherwise, it is a new variable  $x$ , where  $x$  stands for the *lgg* of that pair of terms throughout the computation of the *lgg*. This information is kept in what we call the *lgg* table.

*Example 3.* Let  $C_1 = \{p(a, f(b)), p(g(a, x), c), q(a)\}$  and  $C_2 = \{p(z, f(2)), q(z)\}$ . Their pairs of compatible literals are  $\{p(a, f(b)) - p(z, f(2)), p(g(a, x), c) - p(z, f(2)), q(a) - q(z)\}$ . Their *lgg* is  $\text{lgg}(C_1, C_2) = \{p(X, f(Y)), p(Z, V), q(X)\}$ . The *lgg* table produced during the computation of  $\text{lgg}(C_1, C_2)$  is

[ $a - z \Rightarrow X$ ]	(from $p(\underline{a}, f(b))$ with $p(\underline{z}, f(2))$ )
[ $b - 2 \Rightarrow Y$ ]	(from $p(a, f(\underline{b}))$ with $p(z, f(\underline{2}))$ )
[ $f(b) - f(2) \Rightarrow f(Y)$ ]	(from $p(a, f(\underline{b}))$ with $p(z, f(\underline{2}))$ )
[ $g(a, x) - z \Rightarrow Z$ ]	(from $p(\underline{g(a, x)}, c)$ with $p(\underline{z}, f(2))$ )
[ $c - f(2) \Rightarrow V$ ]	(from $p(\underline{g(a, x)}, \underline{c})$ with $p(z, f(\underline{2}))$ )

The number of literals in the *lgg* of two clauses can be as large as the product of the number of literals in the two clauses and repeated application of *lgg* can lead to an exponential increase in size. Pairings are subsets of the *lgg* that avoid this explosion in size by imposing an additional constraint requiring that each literal in the original clauses is *paired* at most once with a compatible literal of the other clause. In Example 3, we have the literal  $p(z, f(2)) \in C_2$  paired to the

literals  $p(a, f(b))$  and  $p(g(a, x), c)$  of  $C_1$ . A pairing disallows this by including just one copy in the result. Naturally, given two clauses we now have many possible pairings instead of a single  $lgg$ .

Pairings are defined in [13, 3] by way of matchings of terms. Notice that the first two columns of the  $lgg$  table define a matching between terms in the two clauses. In our example this matching is not 1-1 since the term  $f(2)$  in  $C_2$  has been used in more than one entry of the matching, in particular, in entries  $f(b) - f(2)$  and  $c - f(2)$ . This reflects the fact that the atom  $p(z, f(2))$  of  $C_2$  is paired with two atoms in  $C_1$  in the  $lgg$ . Every 1-1 matching corresponding to a 1-1 restriction of the  $lgg$  table induces a pairing.

### 5.1 General Clauses

We first show that general clauses allowing the use of arbitrary terms can have an exponential number of pairings. Fix  $v$  such that  $\log_2 v$  is an integer. Let  $t_{i,j}$  be a ground term that is unique for every pair of integers  $0 \leq i, j \leq v - 1$ . For example,  $t_{i,j}$  could use two unary function symbols  $f_0$  and  $f_1$  and a constant  $a$  and we define  $t_{i,j}$  as a string of applications of  $f_0$  or  $f_1$  of length  $2 \log v$ , finalized with the constant  $a$  such that the first  $\log v$  function symbols encode the binary representation of  $i$  and the last  $\log v$  function symbols encode  $j$ . For example, if  $v = 8$ , then the term  $t_{5,3}$  can be encoded as  $\underbrace{f_1(f_0(f_1(f_0(f_1(f_1(a))))))}_5 \underbrace{f_0(f_1(f_1(a)))}_3$ . The

size of such a term (in terms of symbol occurrences) is exactly  $2 \log v + 1$ . Let  $x_0, \dots, x_{v-1}$  and  $y_0, \dots, y_{v-1}$  be variables. We define

$$C_1 = \bigvee_{\substack{0 \leq i, j < v \\ 0 \leq l < v-1}} p(t_{i,j}, x_l, x_{l+1})$$

$$C_2 = \bigvee_{0 \leq i, j < v} p(t_{i,j}, y_i, y_j).$$

Notice that  $|C_1| = v^2(v - 1)$  and  $|C_2| = v^2$ , and they use a single predicate symbol of arity 3.

Any 1-1 matching between the variables in  $C_1$  and  $C_2$  can be represented by a permutation  $\pi$  of  $\{0, \dots, v - 1\}$ : each variable  $x_i$  in  $C_1$  is matched to  $y_{\pi(i)}$  in  $C_2$ . All the matchings considered in this section map the common ground terms of  $C_1$  and  $C_2$  to one another, i.e., the extended matchings also contain all entries  $[t - t \Rightarrow t]$ , where  $t$  is any ground term appearing in both  $C_1$  and  $C_2$ . Let the extended matching induced by permutation  $\pi$  be

$$\{x_i - y_{\pi(i)} \Rightarrow X_{\pi(i)} \mid 0 \leq i \leq v - 1\} \cup \{t - t \Rightarrow t \mid t \in Terms(C_1) \cap Terms(C_2)\}.$$

First we study  $lgg_\pi(C_1, C_2)$ , the pairing induced by the 1-1 matching represented by  $\pi$ . A literal  $p(t_{i,j}, X_a, X_b)$  is included in  $lgg_\pi(C_1, C_2)$  iff  $a = \pi(l)$  and  $b = \pi(l + 1)$  for some  $l \in \{0, \dots, v - 2\}$  (this is the condition imposed by  $C_1$ ), and  $i = a, j = b$  (this is the condition imposed by  $C_2$ ). Therefore,  $lgg_\pi(C_1, C_2) = \bigvee_{0 \leq l < v-1} p(t_{\pi(l), \pi(l+1)}, X_{\pi(l)}, X_{\pi(l+1)})$ .

Finally we see that different permutations yield pairings that are subsumption inequivalent, i.e.,  $lgg_\pi(C_1, C_2) \not\leq lgg_{\pi'}(C_1, C_2)$  for any  $\pi \neq \pi'$ . It is sufficient to observe that since  $\pi$  and  $\pi'$  are distinct, there must exist some term  $t_{\pi(l), \pi(l+1)}$  in  $lgg_\pi(C_1, C_2)$  that is not present in  $lgg_{\pi'}(C_1, C_2)$ . This holds since a distinct pair of consecutive indices exists for any two permutations. Since the terms  $t_{*,*}$  are ground, subsumption is not possible. There are  $v!$  distinct permutations of  $\{0, \dots, v-1\}$  and therefore:

**Theorem 5.** *Let  $\mathcal{S}$  be a signature containing a predicate symbol of arity at least 3, two unary function symbols and a constant. The number of distinct pairings between a pair of  $\mathcal{S}$ -clauses using  $v$  variables,  $O(v^3)$  literals and terms of size  $O(\log v)$  can be  $\Omega(v!)$ .*

## 5.2 Function Free Clauses

We next generalize the construction to use function free clauses. We start with a construction using non-fixed arity. Our construction mimics the behavior of pairing ground terms in the previous section by using 2 additional variables,  $z_0$  and  $z_1$ , that encode the integers  $i$  and  $j$  in a similar way to  $t_{i,j}$ . By looking at matchings  $\pi$  that match the variables  $z_0$  and  $z_1$  to themselves, we guarantee that the resulting  $lgg_\pi$  contains the correct encoding of the variables in the last and previous-to-last positions of the atoms. Let

$$C_1 = \bigvee_{\substack{(i_1, \dots, i_{\log v}) \in \{0,1\}^{\log v} \\ (j_1, \dots, j_{\log v}) \in \{0,1\}^{\log v} \\ 0 \leq l < v-1}} p(z_{i_1}, \dots, z_{i_{\log v}}, z_{j_1}, \dots, z_{j_{\log v}}, x_l, x_{l+1})$$

$$C_2 = \bigvee_{\substack{(i_1, \dots, i_{\log v}) = \text{binary}(i) \\ (j_1, \dots, j_{\log v}) = \text{binary}(j) \\ 0 \leq i, j < v}} p(z_{i_1}, \dots, z_{i_{\log v}}, z_{j_1}, \dots, z_{j_{\log v}}, y_i, y_j)$$

where we use  $\text{binary}(n)$  to denote the tuple  $z_{n_1}, \dots, z_{n_{\log v}}$  encoding  $n$  in its binary representation using  $z_0, z_1$ . For example, assuming  $v = 8$ ,  $\text{binary}(6) = z_1, z_1, z_0$ . Notice that  $|C_1| = v^2(v-1)$  and  $|C_2| = v^2$ , the clauses use a single predicate symbol of arity  $2 \log v + 2$ , and both clauses use exactly  $v + 2$  variables.

Any 1-1 matching between the variables  $x_0, \dots, x_{v-1}$  in  $C_1$  and  $y_0, \dots, y_{v-1}$  in  $C_2$  can be represented by a permutation  $\pi$  of  $\{0, \dots, v-1\}$ : each variable  $x_i$  in  $C_1$  is matched to  $y_{\pi(i)}$  in  $C_2$ . Let the matching induced by permutation  $\pi$  be

$$\{x_i - y_{\pi(i)} \Rightarrow X_{\pi(i)} \mid 0 \leq i < v\}.$$

First we study  $lgg_{\pi \cup \{z_0 \rightarrow z_0, z_1 \rightarrow z_1\}}(C_1, C_2)$ , the pairing induced by the 1-1 matching represented by  $\pi$  augmented with  $z_0$  and  $z_1$  matched to themselves. A literal  $p(z_{i_1}, \dots, z_{i_{\log v}}, z_{j_1}, \dots, z_{j_{\log v}}, X_a, X_b)$  is included in  $lgg_\pi(C_1, C_2)$  iff  $a = \pi(l)$  and  $b = \pi(l+1)$  for some  $l \in \{0, \dots, v-2\}$  (this is the condition imposed by  $C_1$ ), and  $(i_1, \dots, i_{\log v}) = \text{binary}(a)$ ,  $(j_1, \dots, j_{\log v}) = \text{binary}(b)$

(this is the condition imposed by  $C_2$ ). Therefore,  $lgg_{\pi \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2) = \bigvee_{0 \leq l < v-1} p(\text{binary}(\pi(l)), \text{binary}(\pi(l+1)), X_{\pi(l)}, X_{\pi(l+1)})$ .

Finally, we want to check whether different permutations yield pairings that are subsumption inequivalent, i.e., if for any  $\pi \neq \pi'$

$$lgg_{\pi \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2) \not\subseteq lgg_{\pi' \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2).$$

To this end, we investigate which substitutions  $\theta$  satisfy

$$lgg_{\pi \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2) \cdot \theta \subseteq lgg_{\pi' \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2).$$

If  $\theta$  does not change the values of  $z_0, z_1$ , then as before some atom

$$p(\text{binary}(\pi(l)), \text{binary}(\pi(l+1)), *, *) \cdot \theta = p(\text{binary}(\pi(l)), \text{binary}(\pi(l+1)), *, *)$$

in  $lgg_{\pi \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2) \cdot \theta$  does not occur in  $lgg_{\pi' \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2)$ . If  $\theta$  maps both variables  $z_0, z_1$  to the same value (either  $z_1$  or  $z_0$ ), then inclusion cannot happen since  $lgg_{\pi' \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2)$  contains no atoms of the form  $p(z_0, \dots, z_0, *, *)$  or  $p(z_1, \dots, z_1, *, *)$ . Obviously, if  $z_0$  or  $z_1$  are mapped into any other variable  $X_*$ , then the inclusion is not possible either. Hence,  $\theta$  must exchange the values of  $z_0, z_1$ , and:

$$p(\text{binary}(\pi(l)), \text{binary}(\pi(l+1)), *, *) \cdot \theta = p(\overline{\text{binary}(\pi(l))}, \overline{\text{binary}(\pi(l+1))}, *, *)$$

where  $\overline{\text{binary}(n)}$  is the ‘‘complement’’ of  $\text{binary}(n)$ . For example, assuming  $v = 8$ ,  $\text{binary}(6) = z_0, z_0, z_1$ . More precisely,  $\text{binary}(n) = \text{binary}(v-1-n)$ . We have seen that there is only one permutation  $\pi' = \bar{\pi}$  for which there exists some  $\theta$  s.t.  $lgg_{\pi \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2) \cdot \theta \subseteq lgg_{\pi' \cup \{z_0-z_0, z_1-z_1\}}(C_1, C_2)$ . Moreover,  $\theta$  is exactly  $\{z_0 \mapsto z_1, z_1 \mapsto z_0\} \cup \{X_l \mapsto X_{v-1-l} \mid 0 \leq l < v\}$ . We therefore get:

**Theorem 6.** *Let  $\mathcal{S}$  be a signature containing a predicate symbol of arity at least  $2 \log v + 2$ . The number of distinct pairings between a pair of function free  $\mathcal{S}$ -clauses using  $v + 2$  variables,  $O(v^3)$  literals can be  $\Omega(v!)$ .*

As in the previous section we can generalize the result to use predicates with fixed arity:

**Theorem 7.** *Let  $\mathcal{S}$  be a signature containing a predicate symbol of arity at least 3. The number of distinct pairings between a pair of function free  $\mathcal{S}$ -clauses using at most  $v$  variables and  $v$  literals can be  $\Omega(2^{v/4})$ .*

### 5.3 Implications for Learnability

The Algorithms in [13, 3] are shown to learn first order classes from equivalence and membership queries. The algorithms use pairings in the process of learning and a  $t^v$  upper bound on the number of these is used. No explicit lower bound was given leaving open the possibility that better analysis might yield better upper bounds. The results above can be used to give a concrete example where an exponential number of queries is indeed used. We sketch the details

here for the algorithm in [3]. Let the target be  $T$ . The algorithm maintains a set of meta-clauses as its hypothesis. Two major steps in the algorithm are minimization and pairing. In minimization, given a counter example clause  $C$  s.t.  $T \models C$  the algorithm iterates dropping one object at a time and asking an entailment membership query to check whether it is correct. For example given  $p(x_1, x_2), p(x_2, x_3), p(x_1, x_3), p(x_3, x_4) \rightarrow q(x_3, x_3)$  dropping  $x_2$  (and all atoms using it) yields  $p(x_1, x_3), p(x_3, x_4) \rightarrow q(x_3, x_3)$ . In this way a counter example with a minimal set of variables is obtained. Then the algorithm tries to find a pairing of the minimized example and a meta-clause in the hypothesis which yields an implied clause of smaller size. This is done by enumerating all “basic” pairings. If no such pairing is found then the clause is added as a new meta-clause to the hypothesis. Therefore in order to show that the algorithm makes an exponential number of queries it suffices to show a target  $T = D_1 \wedge D_2$  where (1) each of  $D_1, D_2$  is already minimal so that minimization does not alter them, (2)  $D_1, D_2$  have an exponential number of “basic” pairings, and (3)  $T \not\models C$  for any  $C$  which is a pairing of  $D_1, D_2$ . If this holds then we can give the clause  $D_1$  to the algorithm as a counter example and then follow with  $D_2$ . The algorithm will ask a membership query on all the pairings getting an answer of No every time and eventually add  $D_2$  to its hypothesis. We omit the technical definition of “basic” pairings but note that all pairings constructed in the previous section are “basic” since they map variables to variables.

Let  $f()$  be a nullary predicate symbol, and  $q()$  and  $r()$  binary predicates. Let  $N_1, N_2$  be the number of variables used in  $C_1, C_2$  in the construction above respectively, and rename these variables (in any order) so that  $C_1$  uses variables  $v_1, \dots, v_{N_1}$ , and  $C_2$  uses variables  $w_1, \dots, w_{N_2}$ . Then we use  $q()$  and  $r()$  to define chains of variables touching all variables in  $C_1, C_2$ :  $Q = \bigwedge_{1 \leq i < N_1} q(v_i, v_{i+1})$  and  $R = \bigwedge_{1 \leq i < N_2} r(w_i, w_{i+1})$ . Now define  $C'_1, C'_2$  to be the conjunction of the atoms from  $C_1, C_2$  above (we used disjunction above) and let  $D_1 = C'_1 \wedge Q \rightarrow f()$  and  $D_2 = C'_2 \wedge R \rightarrow f()$ . Finally  $T = D_1 \wedge D_2$ . Since a linear chain formed as in  $Q, R$  never subsumes any subchains and since  $Q, R$  use different predicate symbols we can show the following:

**Lemma 10.** *Let  $T = D_1 \wedge D_2$  as defined above and let  $C$  be any clause.*

- (1)  $D_1 \not\preceq D_2$  and hence  $D_1 \not\models D_2$ .  $D_2 \not\preceq D_1$  and hence  $D_2 \not\models D_1$ .
- (2) If  $T \models C$  then it is the case that either  $D_1 \preceq C$  or  $D_2 \preceq C$ .
- (3) If  $D$  is a result of dropping any object from  $D_1$  or  $D_2$  then  $T \not\models D$ .

**Theorem 8.** *The algorithm of [3] can make  $\Omega(2^{v/4})$  queries on some targets.*

## References

- [1] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.
- [2] Dana Angluin. Queries revisited. In *Proceedings of the International Conference on Algorithmic Learning Theory*, volume 2225 of *Lecture Notes in Computer Science*, pages 12–31, Washington, DC, USA, November 25–28 2001. Springer.

- [3] M. Arias and R. Khardon. Learning closed Horn expressions. *Information and Computation*, 178:214–240, 2002.
- [4] M. Arias and R. Khardon. Complexity parameters for first-order structures. In *Proceedings of the 13th International Conference on Inductive Logic Programming*, pages 22–37. Springer-Verlag, 2003. LNAI 2835.
- [5] M. Arias and R. Khardon. The subsumption lattice and query learning. Technical Report 2004-7, Department of Computer Science, Tufts University, 2004.
- [6] M. Arias, R. Khardon, and R. A. Servedio. Polynomial certificates for propositional classes. In *Proceedings of the Conference on Computational Learning Theory*, pages 537–551. Springer-Verlag, 2003. LNAI 2777.
- [7] Hiroki Arimura. Learning acyclic first-order Horn sentences from entailment. In *Proceedings of the International Conference on Algorithmic Learning Theory*, Sendai, Japan, 1997. Springer-Verlag. LNAI 1316.
- [8] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the 6th Conference on Algorithmic Learning Theory*, volume 997. Springer-Verlag, 1995.
- [9] F. Esposito, N. Fanizzi, S. Ferilli, and G. Semeraro. Ideal theory refinement under object identity. In *Proceedings of the International Conference on Machine Learning*, pages 263–270, 2000.
- [10] Sally A. Goldman and Michael Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50:20–31, 1995.
- [11] T. Hegedus. On generalized teaching dimensions and the query complexity of learning. In *Proceedings of the Conference on Computational Learning Theory*, pages 108–117, New York, NY, USA, July 1995. ACM Press.
- [12] L. Hellerstein, K. Pillaipakkammatt, V. Raghavan, and D. Wilkins. How many queries are needed to learn? *Journal of the ACM*, 43(5):840–862, September 1996.
- [13] R. Khardon. Learning function free Horn expressions. *Machine Learning*, 37:241–275, 1999.
- [14] J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.)*. Springer-Verlag New York, Inc., 1987.
- [15] S. Nienhuys-Cheng and R. De Wolf. *Foundations of Inductive Logic Programming*. Springer-verlag, 1997. LNAI 1228.
- [16] G. D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.
- [17] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
- [18] K. Rao and A. Sattar. Learning from entailment of logic programs with local variables. In *Proceedings of the International Conference on Algorithmic Learning Theory*, Otzenhausen, Germany, 1998. Springer-verlag. LNAI 1501.
- [19] 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.
- [20] 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.
- [21] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.