TUFTS-CS Technical Report 2004-7

August 2004

The Subsumption Lattice and Query Learning

by

Marta Arias
Dept. of Computer Science
Tufts University
Medford, Massachusetts 02155

Roni Khardon
Dept. of Computer Science
Tufts University
Medford, Massachusetts 02155

The Subsumption Lattice and Query Learning*

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

July 22, 2004

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 [16]). 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 has been partly supported by NSF Grant IIS-0099446

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" [20] the query complexity is either exponential in one of the crucial parameters (e.g. the number of universally quantified variables) [14, 3] or the algorithms use additional syntax-based oracles [7, 21, 19]. 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 $c_1 \prec c_2 \prec \ldots \prec c_n$ of first order clauses of restricted size. This is motivated by two issues. First, many ILP algorithms (e.g. [22, 18, 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 [13, 12] to study query complexity. It is known [13, 12] 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 [14, 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 [14, 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 [14, 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 [15, 16]. 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 $x_1, x_2, x_3, \ldots$ is used to construct expressions. A variable is a *term*. If $t_1, \ldots, t_n$ are terms and $f \in F$ is a function symbol of arity n, then $f(t_1, ..., t_n)$ is a term. An *atom* is an expression $p(t_1, ..., t_n)$ where $p \in P$ is a predicate symbol of arity n and $t_1, ..., t_n$ 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 $\neg p_1 \vee ... \vee \neg p_n \vee p_{n+1}$ is equivalent to its implicational form $p_1 \wedge ... \wedge p_n \rightarrow p_{n+1}$. We call $p_1 \wedge ... \wedge p_n$ 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 θ 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 $c_1 \prec c_2 \prec .. \prec c_n$. It is known that infinite chains exist if one does not restrict clause size [17, 16] 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 θ 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. $t \cdot \theta = t' \cdot \theta$. The following two lemmas relate subsumption and size parameters:

Lemma 3.1 *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) $NLiterals(c_1) \leq NLiterals(c_2)$.*

Proof: Let θ be the witnessing substitution for the fact that $c_1 \preceq c_2$. Suppose that θ is unifying w.r.t. c_1. That is, there exist two distinct terms t, t' in c_1 that have been unified and therefore $t \cdot \theta = t' \cdot \theta = \hat{t}$. Since c_1 is fully inequated, the inequality $(t \neq t') \in c_1$. But then $(t \neq t') \cdot \theta$ is precisely $(\hat{t} \neq \hat{t})$ and since c_2 is fully-inequated it cannot be included, contradicting the fact that $c_1 \cdot \theta \subseteq c_2$. Thus (1) holds.

For (2) note that by (1) all distinct terms in c_1 remain distinct in $c_1 \cdot \theta$ because θ is non-unifying. Hence, c_2 has at least as many terms as c_1 since it contains $c_1\theta$. Moreover, θ might replace (light) variables by (heavier) functional terms, and (2) follows.

For (3) note that if $NLiterals(c_1) > NLiterals(c_2)$, then at least two literals in c_1, and hence two terms in c_1, must be unified in c_1, contradicting (1). ■

Lemma 3.2 *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 θ be the substitution such that $c_1\theta \subseteq c_2$. Then θ induces a 1-1 mapping of terms. Now if θ maps a variable to a non-variable term then $WTerms(c_1) < WTerms(c_2)$. So θ must be a variable renaming. If θ 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 $c_2 \not\preceq c_1$. ■

As a result each step in a strict subsumption chain reduces one of *NLiterals* or *WTerms* and we get:

Theorem 3.3 *The longest proper subsumption chain of fully inequated clauses with at most t terms and l literals is of length at most $2t + l$.*

Proof: Let $c_1 \prec c_2 \prec .. \prec c_n$ be a chain of maximal length. By Lemma 3.2, after each step in the chain (from left to right), either we increase the number of literals, or the quantity *WTerms* increases. By Lemmas 3.1 these quantities never decrease. The bound t on the number of terms implies that *WTerms* can never grow beyond $2t$ (in the case that all the terms are functional). Since *NLiterals* cannot surpass l, the number of total clauses in our chain is 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 $d_1 \succ d_2 \succ .. \succ d_n$ is defined inductively. The first clause is $d_1 = p(z,..,z)$, and given a clause $d_i = p_1, p_2, .., p_k$, we define the next clause d_{i+1} as follows: (1) if p_1 contains only two occurrences of the variable z, then $d_{i+1} = p_2, .., p_k$, 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 $p'_1, .., p'_{k'}$ 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 3.1 *Suppose p has arity 4 and that $l = 3$. The construction produces the following chain of length 11:*

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

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:

- $N(2,s) = 1$, 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.
- $N(c,0) = c-1$, 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.
- $N(c,s) = 1 + \sum_{i=\max(0,s-c+1)}^{s} N(c-1,i)$, 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.

Lemma 3.4 *$N(c, s) \geq \binom{c}{s+1} - 1$ for $c \geq 2$ and $s \geq 0$.*

Proof: Recall that in case that $n < k$, $\binom{n}{k} = 0$. The proof is by induction on c, s. The base cases are when $s = 0$ or $c = 2$:

- $N(c, 0) = c - 1 \geq \binom{c}{1} - 1 = c - 1$ for all $c \geq 2$.
- $N(2, s) = 1 \geq \binom{2}{s+1} - 1$ for all $s \geq 0$.

For the step case, assume that $N(c', s') \geq \binom{c'}{s'+1} - 1$ for values $c' < c$ or $s' < s$. Then, if $c \geq 3$ and $s \geq 1$ we have that:

$$
\begin{aligned}
N(c, s) &= 1 + \sum_{i=\max(0, s-c+1)}^{s} N(c-1, i) && (1)\\
&\geq 1 + N(c-1, s) + N(c-1, s-1) && (2)\\
&\geq 1 + \binom{c-1}{s+1} - 1 + \binom{c-1}{s} - 1 && (3)\\
&= \binom{c}{s+1} - 1 && (4)
\end{aligned}
$$

For (2), notice that $c \geq 3$ and $s \geq 1$ imply that $0 \leq s-1$ and $s-c+1 \leq s-1$, hence $\max\{0, s-c+1\} \leq s-1$. For (3) we apply the induction hypothesis, and for (4) we use the basic identity $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ which also holds for n, k such that $k > n$. ■

It remains to show that this is a proper chain. First, we investigate key structural properties of the clauses participating in our chain. It is easy to verify the following lemma by induction on the updates of d_i.

Lemma 3.5 *Let $Vars(p)$ be the variables occurring in the atom p. For all $d_i = p_1, \ldots, p_k$ the following properties hold:*

- *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.*
- *$Vars(p_j) \supseteq Vars(p_{j+1})$ for all $j = 1, \ldots, k-1$.*

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 $B_1, B_2, \ldots, B_m$ such that all the atoms in a single block contain exactly the same variables, and variables appearing in neighboring blocks are such that $Vars(B_j) \supset Vars(B_{j+1})$.

Lemma 3.6 *Fix some clause d_i, and let p be an atom in any block B. If $p \cdot \theta \in B$, then θ does not change variables in p.*

Proof: By induction on the updates of d_i. The claim is trivially true for d_1 since it contains a single atom. For the step case, assume the lemma is true for $d_i = p_1, .., p_k$.

If $d_{i+1} = p_2, .., p_k$ (left-most atom was dropped), then the induction hypothesis guarantees the result. Otherwise, $d_{i+1} = p'_1, .., p'_{k'}, p_2, .., p_k$ (left-most atom replaced by new set containing one more variable in different places). By the induction hypothesis we only have to check that the claim is true for the new block $B = p'_1, .., p'_{k'}$. If $k' = 1$ then B contains a single atom and the lemma is trivially true. Otherwise, B contains at least two atoms. Notice that the way the atoms $p'_1, .., p'_{k'}$ have been created is by replacing the variable z in p_1 by a new variable x, but in different positions in each new atom $p(t'_j)$. Hence, it holds that for every pair $p_1, p_2 \in B$, they agree on all positions except in two where one has the variable z and the other one has the new variable x (and vice versa for the other position). If $p_1 \cdot \theta = p_2$ then θ would have to map the variable z into the newly introduced variable x. But this would result in an atom with at least two occurrences of x, and such atoms do not appear in the clauses we create. Hence, the new variable must be left untouched by θ and therefore there is no θ such that $p_1 \cdot \theta = p_2$. Since p_1, p_2 are arbitrary atoms we conclude that $p \cdot \theta \notin B$ unless $p \cdot \theta = p$ and therefore θ does not change the value of variables in p. ■

Lemma 3.7 *Let d_i be any clause in the sequence and let $B_1, .., B_m$ 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 $Vars(B_{i_2}) \setminus \{z\}$ 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.*

Proof: By induction on the updates of d_i. The claim is trivially true for d_1 since it contains a single atom and hence a single block. For the step case, assume the lemma is true for $d_i = p_1, .., p_k$.

If $d_{i+1} = p_2, .., p_k$ (left-most atom was dropped), then the induction hypothesis guarantees the result. If $d_{i+1} = p'_1, .., p'_{k'}, p_2, .., p_k$, then the property is guaranteed by the induction hypothesis for pairs of blocks in $p_2, .., p_k$. It remains to check that the lemma is true when $B_{i_1} = p'_1, .., p'_{k'}$ and B_{i_2} is any other block in d_{i+1}. If the replaced atom $p_1 \in d_i$ appeared in a different block in d_i as the atoms in B_{i_2}, then the induction hypothesis applies. If p_1 appeared in the same block as the atoms in B_{i_2}, then the variable that was introduced by the creation of that block has to be in different positions in all the atoms in B_{i_2}. Since all the atoms in $p'_1, .., p'_{k'}$ inherit this variable from p_1, the lemma follows. ■

Lemma 3.8 *Fix some clause $d_i = p_1, .., p_k$ with at least 2 atoms (i.e., $k \geq 2$). Then $(p_2, .., p_k) \cdot \theta \subseteq d_i$ only if θ does not change variables in p_2.*

Proof: Let $d_i = B_1, .., B_m$. Let p be any atom in any block B_j and assume $p \cdot \theta \in d_i$. Notice that $p \cdot \theta \notin B_1, .., B_{j-1}$ since atoms in blocks $B_1, .., B_{j-1}$ contain strictly more variables than $p \cdot \theta$. Hence $p \cdot \theta \in B_j, .., B_m$. We first claim that if θ does not change variables in $B_{j+1}, .., B_m$ then θ does not change variables in B_j. To prove the claim note

that if $p \cdot \theta \in B_j$ then Lemma 3.6 applies. On the other hand if $p \cdot \theta \in B_{j'}$ for $j < j'$ then Lemma 3.7 guarantees that there exists some variable $x \in Vars(B_{j'})$ that appears in a position in p in which atoms in $B_{j'}$ contain the variable z. Since by assumption θ does not change this variable this implies that $p \cdot \theta \notin B_{j'}$ leading to a contradiction.

Finally if $(p_2, .., p_k) \cdot \theta \subseteq d_i$ 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 θ does not change variables that appear in the leftmost block of $p_2, .., p_k$, and hence in p_2 as required. ■

Lemma 3.9 *For all $i = 1, .., n-1$ we have that $d_i \succ d_{i+1}$.*

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

Case 1. $d_{i+1} = p_2, .., p_k$. 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 θ 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 $(p_2 \; .., p_k) \cdot \theta \subseteq d_i$, and by Lemma 3.8, θ 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 3.7 guarantees that for every atom in $p_2, .., p_k$ there is a variable that appears in a different location in p_1 and as above this variable cannot be changed by θ. Hence, $p_1 \cdot \theta \notin d_{i+1}$, contradicting our assumption that $d_i \preceq d_{i+1}$.

Case 2. $d_{i+1} = p'_1, .., p'_{k'}, p_2, .., p_k$. 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 θ such that $d_i \cdot \theta \subseteq d_{i+1}$. If $d_i = p_1$, (i.e., d_i contains one atom only), then $p_1 \cdot \theta \subseteq p'_1, .., p'_{k'}$. In this case, θ must map z into the new variable x but this results in multiple occurrences of x, and hence $p_1 \cdot \theta \not\subseteq p'_1, .., p'_{k'}$. Hence, d_i must contain at least two atoms and the substitution θ must satisfy that $(p_1, .., p_k) \cdot \theta \subseteq p'_1, .., p'_{k'}, p_2, .., p_k$. The new atoms $p'_1, .., p'_{k'}$ contain more variables than $p_1, .., p_k$, therefore $(p_1, .., p_k) \cdot \theta \subseteq p_2, .., p_k$. By the same reasoning as in the previous case, we conclude that $d_i \succ d_{i+1}$. ■

Now Lemma 3.4 and Lemma 3.9 imply:

Theorem 3.10 *Let p be a predicate symbol of arity $\alpha \geq 1$. There exists a proper subsumption chain of length $\binom{\alpha}{l}$ of function free clauses using at most α variables and l literals.*

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

Definition 3.1 *Let d be any clause. Let $Trans(d)$ be the clause obtained by replacing each literal $p(t_1, .., t_a)$ with a new set $\{p(y_i, y_{i+1}, t_i) \mid 1 \leq i \leq a\}$, where all $y_1, .., y_{a+1}$ are new variables not appearing in d. The new variables $y_1, .., y_{a+1}$ are different for each atom in d.*

Example 3.2 *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 3.11 *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 θ from variables in d_1 into terms of d_2 such that $d_1 \cdot \theta \subseteq d_2$. Obviously, θ 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 θ such that $Trans(d_1)\cdot\theta \subseteq Trans(d_2)$. Let $d_1 = l_1^1 \vee l_1^2 \vee .. \vee l_1^{k_1}$ and let $\{y_1^j, .., y^j_{arity(l_1^j)+1}\}$ be the variables used in the transformation for literal l_1^j in d_1, for $1 \leq j \leq k_1$. Similarly, let $d_2 = l_2^1 \vee l_2^2 \vee .. \vee l_2^{k_2}$ and let $\{y'^j_1, .., y'^j_{arity(l_2^j)+1}\}$ 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 θ must map blocks of auxiliary variables in $Trans(d_1)$, $\{y_1^j, .., y^j_{arity(l_1^j)+1}\}$ into blocks of auxiliary variables in $Trans(d_2)$, $\{y'^{j'}_1, .., y'^{j'}_{arity(l_2^{j'})+1}\}$ 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., θ maps each $y_i^j \mapsto y'^{j'}_i$, for all $1 \leq i \leq arity(l_2^{j'})$. By way of contradiction, suppose that there exists a pair of variables in $Trans(d_1)$, y_i^j and y_{i+1}^j, that have been mapped into y'^a_* and y'^b_*, respectively, where $a \neq b$. Then, $p(y_i^j, y_{i+1}^j, *)\cdot\theta = p(y'^a_*, y'^b_*, *) \in Trans(d_2)$. This contradicts the fact that, by construction, all literals in $Trans(d_2)$ are such that the superscripts of the first two auxiliary variables coincide.

Suppose now that some y_i^j has been mapped into $y'^{j'}_{i'}$ 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 $p(y_{i-1}^j, y_i^j, *)\cdot\theta = p(y'^{j'}_{i-1}, y'^{j'}_{i'}, *) \in Trans(d_2)$. But this is a contradiction since all literals in $Trans(d_2)$ 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. θ maps $y^j_{i+h} \mapsto y'^{j'}_{i'+h}$ but θ does not map $y^j_{i+h+1} \mapsto y'^{j'}_{i'+h+1}$. Thus we arrive to the same contradiction as in the previous case.

Now, the fact that each $y_i^j \mapsto y'^{j'}_i$ implies that θ 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 $d_1 \cdot \theta \subseteq d_2$. ■

Theorem 3.12 *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 3.10 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 $Trans(d_1) \succ Trans(d_2) \succ .. \succ Trans(d_n)$. Lemma 3.11 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. ■

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 4.1 *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 $\langle t_1, .., t_k \rangle$ denotes the term represented by a complete binary tree of applications of a binary function symbol f of depth $\log k$ with leaves $t_1, .., t_k$. 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 $\langle t_1, .., t_k \rangle$ 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 $NTerms(\langle t_1, .., t_k \rangle) \leq 2k$.

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

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

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

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 $\mathcal{C} = \{$first-order monotone $\mathcal{S}$-clauses with at most l atoms$\}$ and the set of examples be $\mathcal{E} = \{$first order ground monotone $\mathcal{S}$-clauses with at most l 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 $f = p(\langle a, .., a\rangle)$ 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 $K = \binom{\binom{k}{k/2}}{l} > (\frac{2^{k/2}}{l})^l = 2^{\Omega(lk)}$ such subsets where we use an additional restriction that $l \leq k$. Let these be $C^1_{k/2}, .., C^K_{k/2}$. By definition, the teaching set T has to reject each one of these K clauses.

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

Theorem 4.1 *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 [17] (see also [16]) 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. We proceed with formal definitions.

Definition 5.1 *A pair of literals are* compatible *if they use the same predicate symbol (and hence same arity) and have the same sign. A pair of first-order terms are* compatible *if they agree on their leftmost function symbol (and hence on their arity as well).*

The algorithm computing the *lgg* is as follows:

LGG(C_1, C_2)

1 **if** C_1, C_2 are clauses
2 **then** $S \leftarrow \emptyset$
3 **for** each pair of compatible literals $l_1 \in C_1$ and $l_2 \in C_2$
4 **do** $S \leftarrow S \cup$ LGG(l_1, l_2)
5 **return** S
6 **if** C_1, C_2 are compatible literals
7 **then if** $C_1 = p(t_1 \ldots t_n)$, $C_2 = p(t'_1 \ldots t'_n)$ are compatible positive literals
8 **then return** $p($LGG$(t_1, t'_1) \ldots$ LGG$(t_n, t'_n))$
9 **else** $/*$ $C_1 = \neg p(t_1 \ldots t_n)$ and $C_2 = \neg p(t'_1 \ldots t'_n)$ $*/$
10 **return** $\neg p($LGG$(t_1, t'_1) \ldots$ LGG$(t_n, t'_n))$
11 **if** C_1, C_2 are first-order terms
12 **then if** $C_1 = f(t_1 \ldots t_n)$, $C_2 = f(t'_1 \ldots t'_n)$ are compatible terms
13 **then return** $f($LGG$(t_1, t'_1) \ldots$ LGG$(t_n, t'_n))$
14 **else return** a new variable x

This procedure is designed to be initially called with two clauses as arguments; in the subsequent recursive calls the arguments are either compatible literals of first-order terms.

It is important to note that whenever the *lgg* returns a new variable (step 14 in LGG) the algorithm stores the fact that the pair C_1, C_2 has been mapped to x into what we call the *lgg table*. If this pair of terms come up again, they are mapped to the same variable. More formally, the *lgg* table produced by the computation of $lgg(C_1, C_2)$ is a mapping from $Terms(C_1) \times Terms(C_2)$ into the new set of terms $Terms(lgg(C_1, C_2))$. We denote the *lgg* tables as sets of ordered triplets of the form `[`t_1 `-` t_2 `=>` t_3`]`, meaning that t_1 and t_2 are mapped to $t_3 = lgg(t_1, t_2)$.

Example 5.1 *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* $lgg(C_1, C_2) = \{p(X, f(Y)), p(Z, V), q(X)\}$. *The* lgg table *produced during the computation of* $lgg(C_1, C_2)$ *is*

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

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 5.1, 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 [14, 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 $\underbrace{f_1(f_0(f_1}_{5}\underbrace{(f_0(f_1(f_1}_{3}(a))))))$. The size of such a term (in terms of symbol occurrences) is exactly $2 \log v + 1$. Let $x_0, \ldots, x_{v-1}$ and $y_0, \ldots, y_{v-1}$ 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 π of $\{0, \ldots, v-1\}$: 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 `=>` t`]`, where t is any ground term appearing in both C_1 and C_2. Let the extended matching induced by permutation π be

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

First we study $lgg_\pi(C_1, C_2)$, the pairing induced by the 1-1 matching represented by π. 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 $l \in \{0, \ldots, v-2\}$ (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., $lgg_\pi(C_1, C_2) \not\preceq lgg_{\pi'}(C_1, C_2)$ for any $\pi \neq \pi'$. It is sufficient to observe that since π and π' 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 $\{0, \ldots, v-1\}$ and therefore:

Theorem 5.1 *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 π that match the variables z_0 and z_1 to themselves, we guarantee that the resulting lgg_π contains the correct encoding of the variables in the last and previous-to-last positions of the atoms. Let

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

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

where we use $binary(n)$ to denote the tuple $z_{n_1},..,z_{n_{\log v}}$ encoding n in its binary representation using z_0, z_1. For example, assuming $v = 8$, $binary(6) = z_1, z_1, z_0$. 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 $x_0,..,x_{v-1}$ in C_1 and $y_0,..,y_{v-1}$ in C_2 can be represented by a permutation π of $\{0,..,v-1\}$: each variable x_i in C_1 is matched to $y_{\pi(i)}$ in C_2. Let the matching induced by permutation π be

$$\left\{x_i - y_{\pi(i)} \Rightarrow X_{\pi(i)} \;\middle|\; 0 \le i < v\right\}.$$

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

Example 5.2 *Let $v = 4$ and let $\pi = (3201)$. Hence, in this example we use a predicate symbol p of arity 6. For clarity, we omit the predicate symbol throughout the example and denote atom $p(t_1,..,t_6)$ by just the argument tuple $(t_1,..,t_6)$. Also, we omit the disjunction operator $\vee$.*

Then, clause C_1 is

$$(z_0,z_0,z_0,z_0,x_0,x_1)\ (z_0,z_0,z_0,z_1,x_0,x_1)\ (z_0,z_0,z_1,z_0,x_0,x_1)\ (z_0,z_0,z_1,z_1,x_0,x_1)$$
$$(z_0,z_1,z_0,z_0,x_0,x_1)\ (z_0,z_1,z_0,z_1,x_0,x_1)\ (z_0,z_1,z_1,z_0,x_0,x_1)\ (z_0,z_1,z_1,z_1,x_0,x_1)$$
$$(z_1,z_0,z_0,z_0,x_0,x_1)\ (z_1,z_0,z_0,z_1,x_0,x_1)\ \underline{(z_1,z_0,z_1,z_0,x_0,x_1)}\ (z_1,z_0,z_1,z_1,x_0,x_1)$$
$$(z_1,z_1,z_0,z_0,x_0,x_1)\ (z_1,z_1,z_0,z_1,x_0,x_1)\ \underline{(z_1,z_1,z_1,z_0,x_0,x_1)}\ (z_1,z_1,z_1,z_1,x_0,x_1)$$

$$(z_0,z_0,z_0,z_0,x_1,x_2)\ (z_0,z_0,z_0,z_1,x_1,x_2)\ (z_0,z_0,z_1,z_0,x_1,x_2)\ (z_0,z_0,z_1,z_1,x_1,x_2)$$
$$\underline{(z_0,z_1,z_0,z_0,x_1,x_2)}\ (z_0,z_1,z_0,z_1,x_1,x_2)\ (z_0,z_1,z_1,z_0,x_1,x_2)\ (z_0,z_1,z_1,z_1,x_1,x_2)$$
$$\underline{(z_1,z_0,z_0,z_0,x_1,x_2)}\ (z_1,z_0,z_0,z_1,x_1,x_2)\ (z_1,z_0,z_1,z_0,x_1,x_2)\ (z_1,z_0,z_1,z_1,x_1,x_2)$$
$$(z_1,z_1,z_0,z_0,x_1,x_2)\ (z_1,z_1,z_0,z_1,x_1,x_2)\ (z_1,z_1,z_1,z_0,x_1,x_2)\ (z_1,z_1,z_1,z_1,x_1,x_2)$$

$$(z_0,z_0,z_0,z_0,x_2,x_3)\ \overline{(z_0,z_0,z_0,z_1,x_2,x_3)}\ (z_0,z_0,z_1,z_0,x_2,x_3)\ (z_0,z_0,z_1,z_1,x_2,x_3)$$
$$(z_0,z_1,z_0,z_0,x_2,x_3)\ (z_0,z_1,z_0,z_1,x_2,x_3)\ (z_0,z_1,z_1,z_0,x_2,x_3)\ (z_0,z_1,z_1,z_1,x_2,x_3)$$
$$(z_1,z_0,z_0,z_0,x_2,x_3)\ (z_1,z_0,z_0,z_1,x_2,x_3)\ (z_1,z_0,z_1,z_0,x_2,x_3)\ (z_1,z_0,z_1,z_1,x_2,x_3)$$
$$(z_1,z_1,z_0,z_0,x_2,x_3)\ (z_1,z_1,z_0,z_1,x_2,x_3)\ (z_1,z_1,z_1,z_0,x_2,x_3)\ (z_1,z_1,z_1,z_1,x_2,x_3)$$

Clause C_2 is

$$(z_0,z_0,z_0,z_0,y_0,y_0)\ \overline{(z_0,z_0,z_0,z_1,y_0,y_1)}\ (z_0,z_0,z_1,z_0,y_0,y_2)\ (z_0,z_0,z_1,z_1,y_0,y_3)$$
$$\underline{(z_0,z_1,z_0,z_0,y_1,y_0)}\ (z_0,z_1,z_0,z_1,y_1,y_1)\ (z_0,z_1,z_1,z_0,y_1,y_2)\ (z_0,z_1,z_1,z_1,y_1,y_3)$$
$$\underline{(z_1,z_0,z_0,z_0,y_2,y_0)}\ (z_1,z_0,z_0,z_1,y_2,y_1)\ (z_1,z_0,z_1,z_0,y_2,y_2)\ (z_1,z_0,z_1,z_1,y_2,y_3)$$
$$(z_1,z_1,z_0,z_0,y_3,y_0)\ (z_1,z_1,z_0,z_1,y_3,y_1)\ \underline{(z_1,z_1,z_1,z_0,y_3,y_2)}\ (z_1,z_1,z_1,z_1,y_3,y_3)$$

The matching induced by $\pi = (3201)$ is

$$\{x_0 - y_3 \Rightarrow X_3, x_1 - y_2 \Rightarrow X_2, x_2 - y_0 \Rightarrow X_0, x_3 - y_1 \Rightarrow X_1\}.$$

And $lgg_{\pi\cup\{z_0-z_0,z_1-z_1\}}(C_1,C_2)$ is (notice that we have marked literals of C_1 and C_2 which participate in this lgg)

$$(z_1,z_1,z_1,z_0,X_3,X_2)\ (z_1,z_0,z_0,z_0,X_2,X_0)\ (z_0,z_0,z_0,z_1,X_0,X_1)$$

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

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

To this end, we investigate which substitutions θ 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 θ does not change the values of z_0, z_1, then as before some atom

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

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 θ 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 $p(z_0, .., z_0, *, *)$ or $p(z_1, .., z_1, *, *)$. Obviously, if z_0 or z_1 are mapped into any other variable X_*, then the inclusion is not possible either. Hence, θ must exchange the values of z_0, z_1, and:

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

where $\overline{binary(n)}$ is the "complement" of $binary(n)$. For example, assuming $v = 8$, $\overline{binary(6)} = z_0, z_0, z_1$. More precisely, $\overline{binary(n)} = binary(v - 1 - n)$. We have seen that there is only one permutation $\pi' = \overline{\pi}$ for which there exists some θ 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, θ 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 5.2 *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 $\Omega(v!)$.*

5.3 Function Free Clauses with Fixed Arity

As in the previous section we can improve the result to use arity 3 predicates by using the construction of Lemma 3.11. Using the same clauses C_1 and C_2 from the previous construction, we establish that for some appropriate 1-1 matching M_π it holds:

$$lgg_{M_\pi}(Trans(C_1), Trans(C_2)) \quad \approx \quad Trans(lgg_{\pi \cup \{z_0 - z_0, z_1 - z_1\}}(C_1, C_2)), \tag{5}$$

where $\approx$ indicates that the clauses are syntactic variants, that is they are the same up to variable renaming.

In the previous section we established that there are $\frac{v!}{2}$ distinct pairings between C_1, C_2. Lemma 3.11 guarantees that the transformation on clauses $Trans(\cdot)$ preserves subsumption, hence there must be also $\frac{v!}{2}$ distinct clauses corresponding to the right hand side of Equation (5). Equation (5) therefore establishes that there are also $\frac{v!}{2}$ different pairings between $Trans(C_1)$ and $Trans(C_2)$. Moreover, the clauses $Trans(C_1)$ and $Trans(C_2)$ use resources within bounds, namely, they use a polynomial number of atoms (in v), a polynomial number of variables (in v), but fixed arity 3.

To fix notation, let us unfold the transformation:

$$Trans(C_1) = \bigvee_{\substack{i=(i_1,..,i_{\log v}) \in \{0,1\}^{\log v} \\ j=(j_1,..,j_{\log v}) \in \{0,1\}^{\log v} \\ 0 \leq l < v-1}} P_{l,i,j,x_l,x_{l+1}}$$

$$Trans(C_2) = \bigvee_{\substack{(i_1,..,i_{\log v}) \ = \ binary(i) \\ (j_1,..,j_{\log v}) \ = \ binary(j) \\ 0 \leq i,j < v}} P_{i,j,y_i,y_j}$$

where

$$\begin{aligned} P_{l,i,j,A,B} &= p(u_1^{l,i,j}, u_2^{l,i,j}, z_{i_1}) \vee \ldots \vee p(u_{\log v}^{l,i,j}, u_{\log v+1}^{l,i,j}, z_{i_{\log v}}) \vee \\ & \quad p(u_{\log v+1}^{l,i,j}, u_{\log v+2}^{l,i,j}, z_{j_1}) \vee \ldots \vee p(u_{2\log v}^{l,i,j}, u_{2\log v+1}^{l,i,j}, z_{j_{\log v}}) \vee \\ & \quad p(u_{2\log v+1}^{l,i,j}, u_{2\log v+2}^{l,i,j}, A) \vee p(u_{2\log v+2}^{l,i,j}, u_{2\log v+3}^{l,i,j}, B), \end{aligned}$$

$$\begin{aligned} P_{i,j,A,B} &= p(w_1^{i,j}, w_2^{i,j}, z_{i_1}) \vee \ldots \vee p(w_{\log v}^{i,j}, w_{\log v+1}^{i,j}, z_{i_{\log v}}) \vee \\ & \quad p(w_{\log v+1}^{i,j}, w_{\log v+2}^{i,j}, z_{j_1}) \vee \ldots \vee p(w_{2\log v}^{i,j}, w_{2\log v+1}^{i,j}, z_{j_{\log v}}) \vee \\ & \quad p(w_{2\log v+1}^{i,j}, w_{2\log v+2}^{i,j}, A) \vee p(w_{2\log v+2}^{i,j}, w_{2\log v+3}^{i,j}, B), \end{aligned}$$

Intuitively, the clause $P_{l,i,j,x_l,x_{l+1}}$ uses the additional variables $\{u_k^{l,i,j}\}_{1\le k\le 2\log v+3}$ to "encode" the atom $p(binary(i), binary(j), x_l, x_{l+1})$ in C_1, i.e.

$$P_{l,i,j,x_l,x_{l+1}} = Trans(p(binary(i), binary(j), x_l, x_{l+1})).$$

Similarly, the clause P_{i,j,y_i,y_j} uses the set of auxiliary variables $\{w_k^{i,j}\}_{1\le k\le 2\log v+3}$ to "encode" the atom $p(binary(i), binary(j), y_i, y_j)$ in C_2, i.e.,

$$P_{i,j,y_i,y_j} = Trans(p(binary(i), binary(j), y_i, y_j)).$$

Notice that $Trans(C_1)$ uses $\Theta(v^3 \log v)$ literals and variables, and $Trans(C_2)$ uses $\Theta(v^2 \log v)$ literals and variables. Both use a single predicate of arity 3.

Example 5.3 *Following Example 5.2, let $p(z_1, z_0, z_1, z_1, x_1, x_2)$ be an atom in C_1 and $p(z_0, z_1, z_1, z_0, y_1, y_2)$ be an atom in C_2. Then,*

$$\begin{aligned} P_{1,2,3,x_1,x_2} &= Trans(p(z_1, z_0, z_1, z_1, x_1, x_2)) \\ &= p(u_1^{1,2,3}, u_2^{1,2,3}, z_1) \vee p(u_2^{1,2,3}, u_3^{1,2,3}, z_0) \\ &\vee \; p(u_3^{1,2,3}, u_4^{1,2,3}, z_1) \vee p(u_4^{1,2,3}, u_5^{1,2,3}, z_1) \\ &\vee \; p(u_5^{1,2,3}, u_6^{1,2,3}, x_1) \vee p(u_6^{1,2,3}, u_7^{1,2,3}, x_2) \end{aligned}$$

$$\begin{aligned} P_{1,2,y_1,y_2} &= Trans(p(z_0, z_1, z_1, z_0, y_1, y_2)) \\ &= p(w_1^{1,2}, w_2^{1,2}, z_0) \vee p(w_2^{1,2}, w_3^{1,2}, z_1) \\ &\vee \; p(w_3^{1,2}, w_4^{1,2}, z_1) \vee p(w_4^{1,2}, w_5^{1,2}, z_1) \\ &\vee \; p(w_5^{1,2}, w_6^{1,2}, y_1) \vee p(w_6^{1,2}, w_7^{1,2}, y_2) \end{aligned}$$

Then $Trans(C_1) =$

$$
\begin{array}{l}
(u_1^{0,0,0},u_2^{0,0,0},z_0)\ (u_2^{0,0,0},u_3^{0,0,0},z_0)\ (u_3^{0,0,0},u_4^{0,0,0},z_0)\ (u_4^{0,0,0},u_5^{0,0,0},z_0)\ (u_5^{0,0,0},u_6^{0,0,0},x_0)\ (u_6^{0,0,0},u_7^{0,0,0},x_1)\\
(u_1^{0,0,1},u_2^{0,0,1},z_0)\ (u_2^{0,0,1},u_3^{0,0,1},z_0)\ (u_3^{0,0,1},u_4^{0,0,1},z_0)\ (u_4^{0,0,1},u_5^{0,0,1},z_1)\ (u_5^{0,0,1},u_6^{0,0,1},x_0)\ (u_6^{0,0,1},u_7^{0,0,1},x_1)\\
\vdots\\
\hline
(u_1^{0,3,2},u_2^{0,3,2},z_1)\ (u_2^{0,3,2},u_3^{0,3,2},z_1)\ (u_3^{0,3,3},u_4^{0,3,3},z_1)\ (u_4^{0,3,2},u_5^{0,3,2},z_0)\ (u_5^{0,3,2},u_6^{0,3,2},x_0)\ (u_6^{0,3,2},u_7^{0,3,2},x_1)\\
(u_1^{0,3,3},u_2^{0,3,3},z_1)\ (u_2^{0,3,3},u_3^{0,3,3},z_1)\ (u_3^{0,3,3},u_4^{0,3,3},z_1)\ (u_4^{0,3,3},u_5^{0,3,3},z_1)\ (u_5^{0,3,3},u_6^{0,3,3},x_0)\ (u_6^{0,3,3},u_7^{0,3,3},x_1)\\
\\
(u_1^{1,0,0},u_2^{1,0,0},z_0)\ (u_2^{1,0,0},u_3^{1,0,0},z_0)\ (u_3^{1,0,0},u_4^{1,0,0},z_0)\ (u_4^{1,0,0},u_5^{1,0,0},z_0)\ (u_5^{1,0,0},u_6^{1,0,0},x_1)\ (u_6^{1,0,0},u_7^{1,0,0},x_2)\\
(u_1^{1,0,1},u_2^{1,0,1},z_0)\ (u_2^{1,0,1},u_3^{1,0,1},z_0)\ (u_3^{1,0,1},u_4^{1,0,1},z_0)\ (u_4^{1,0,1},u_5^{1,0,1},z_1)\ (u_5^{1,0,1},u_6^{1,0,1},x_1)\ (u_6^{1,0,1},u_7^{1,0,1},x_2)\\
\vdots\\
\hline
(u_1^{1,2,0},u_2^{1,2,0},z_1)\ (u_2^{1,2,0},u_3^{1,2,0},z_0)\ (u_3^{1,2,0},u_4^{1,2,0},z_0)\ (u_4^{1,2,0},u_5^{1,2,0},z_0)\ (u_5^{1,2,0},u_6^{1,2,0},x_1)\ (u_6^{1,3,3},u_7^{1,2,0},x_2)\\
\vdots\\
(u_1^{1,3,3},u_2^{1,3,3},z_1)\ (u_2^{1,3,3},u_3^{1,3,3},z_1)\ (u_3^{1,3,3},u_4^{1,3,3},z_1)\ (u_4^{1,3,3},u_5^{1,3,3},z_1)\ (u_5^{1,3,3},u_6^{1,3,3},x_1)\ (u_6^{1,3,3},u_7^{1,3,3},x_2)\\
\\
(u_1^{2,0,0},u_2^{2,0,0},z_0)\ (u_2^{2,0,0},u_3^{2,0,0},z_0)\ (u_3^{2,0,0},u_4^{2,0,0},z_0)\ (u_4^{2,0,0},u_5^{2,0,0},z_0)\ (u_5^{2,0,0},u_6^{2,0,0},x_2)\ (u_6^{2,0,0},u_7^{2,0,0},x_3)\\
\hline
(u_1^{2,0,1},u_2^{2,0,1},z_0)\ (u_2^{2,0,1},u_3^{2,0,1},z_0)\ (u_3^{2,0,1},u_4^{2,0,1},z_0)\ (u_4^{2,0,1},u_5^{2,0,1},z_1)\ (u_5^{2,0,1},u_6^{2,0,1},x_2)\ (u_6^{2,0,1},u_7^{2,0,1},x_3)\\
\vdots\\
(u_1^{2,3,3},u_2^{2,3,3},z_1)\ (u_2^{2,3,3},u_3^{2,3,3},z_1)\ (u_3^{2,3,3},u_4^{2,3,3},z_1)\ (u_4^{2,3,3},u_5^{2,3,3},z_1)\ (u_5^{2,3,3},u_6^{2,3,3},x_2)\ (u_6^{2,3,3},u_7^{2,3,3},x_3)
\end{array}
$$

$Trans(C_2) =$

$$
\begin{array}{l}
(w_1^{0,0},w_2^{0,0},z_0)\ (w_2^{0,0},w_3^{0,0},z_0)\ (w_3^{0,0},w_4^{0,0},z_0)\ (w_4^{0,0},w_5^{0,0},z_0)\ (w_5^{0,0},w_6^{0,0},y_0)\ (w_6^{0,0},w_7^{0,0},y_0)\\
\hline
(w_1^{0,1},w_2^{0,1},z_0)\ (w_2^{0,1},w_3^{0,1},z_0)\ (w_3^{0,1},w_4^{0,1},z_0)\ (w_4^{0,1},w_5^{0,1},z_1)\ (w_5^{0,1},w_6^{0,1},y_0)\ (w_6^{0,1},w_7^{0,1},y_1)\\
(w_1^{0,2},w_2^{0,2},z_0)\ (w_2^{0,2},w_3^{0,2},z_0)\ (w_3^{0,2},w_4^{0,2},z_1)\ (w_4^{0,2},w_5^{0,2},z_0)\ (w_5^{0,2},w_6^{0,2},y_0)\ (w_6^{0,2},w_7^{0,2},y_2)\\
(w_1^{0,3},w_2^{0,3},z_0)\ (w_2^{0,3},w_3^{0,3},z_0)\ (w_3^{0,3},w_4^{0,3},z_1)\ (w_4^{0,3},w_5^{0,3},z_1)\ (w_5^{0,3},w_6^{0,3},y_0)\ (w_6^{0,3},w_7^{0,3},y_3)\\
(w_1^{1,0},w_2^{1,0},z_0)\ (w_2^{1,0},w_3^{1,0},z_1)\ (w_3^{1,0},w_4^{1,0},z_0)\ (w_4^{1,0},w_5^{1,0},z_0)\ (w_5^{1,0},w_6^{1,0},y_1)\ (w_6^{1,0},w_7^{1,0},y_0)\\
(w_1^{1,1},w_2^{1,1},z_0)\ (w_2^{1,1},w_3^{1,1},z_1)\ (w_3^{1,1},w_4^{1,1},z_0)\ (w_4^{1,1},w_5^{1,1},z_1)\ (w_5^{1,1},w_6^{1,1},y_1)\ (w_6^{1,1},w_7^{1,1},y_1)\\
(w_1^{1,2},w_2^{1,2},z_0)\ (w_2^{1,2},w_3^{1,2},z_1)\ (w_3^{1,2},w_4^{1,2},z_1)\ (w_4^{1,2},w_5^{1,2},z_0)\ (w_5^{1,2},w_6^{1,2},y_1)\ (w_6^{1,2},w_7^{1,2},y_2)\\
(w_1^{1,3},w_2^{1,3},z_0)\ (w_2^{1,3},w_3^{1,3},z_1)\ (w_3^{1,3},w_4^{1,3},z_1)\ (w_4^{1,3},w_5^{1,3},z_1)\ (w_5^{1,3},w_6^{1,3},y_1)\ (w_6^{1,3},w_7^{1,3},y_3)\\
\hline
(w_1^{2,0},w_2^{2,0},z_1)\ (w_2^{2,0},w_3^{2,0},z_0)\ (w_3^{2,0},w_4^{2,0},z_0)\ (w_4^{2,0},w_5^{2,0},z_0)\ (w_5^{2,0},w_6^{2,0},y_2)\ (w_6^{2,0},w_7^{2,0},y_0)\\
(w_1^{2,1},w_2^{2,1},z_1)\ (w_2^{2,1},w_3^{2,1},z_0)\ (w_3^{2,1},w_4^{2,1},z_0)\ (w_4^{2,1},w_5^{2,1},z_1)\ (w_5^{2,1},w_6^{2,1},y_2)\ (w_6^{2,1},w_7^{2,1},y_1)\\
(w_1^{2,2},w_2^{2,2},z_1)\ (w_2^{2,2},w_3^{2,2},z_0)\ (w_3^{2,2},w_4^{2,2},z_1)\ (w_4^{2,2},w_5^{2,2},z_0)\ (w_5^{2,2},w_6^{2,2},y_2)\ (w_6^{2,2},w_7^{2,2},y_2)\\
(w_1^{2,3},w_2^{2,3},z_1)\ (w_2^{2,3},w_3^{2,3},z_0)\ (w_3^{2,3},w_4^{2,3},z_1)\ (w_4^{2,3},w_5^{2,3},z_1)\ (w_5^{2,3},w_6^{2,3},y_2)\ (w_6^{2,3},w_7^{2,3},y_3)\\
(w_1^{3,0},w_2^{3,0},z_1)\ (w_2^{3,0},w_3^{3,0},z_1)\ (w_3^{3,0},w_4^{3,0},z_0)\ (w_4^{3,0},w_5^{3,0},z_0)\ (w_5^{3,0},w_6^{3,0},y_3)\ (w_6^{3,0},w_7^{3,0},y_0)\\
(w_1^{3,1},w_2^{3,1},z_1)\ (w_2^{3,1},w_3^{3,1},z_1)\ (w_3^{3,1},w_4^{3,1},z_0)\ (w_4^{3,1},w_5^{3,1},z_1)\ (w_5^{3,1},w_6^{3,1},y_3)\ (w_6^{3,1},w_7^{3,1},y_1)\\
\hline
(w_1^{3,2},w_2^{3,2},z_1)\ (w_2^{3,2},w_3^{3,2},z_1)\ (w_3^{3,2},w_4^{3,2},z_1)\ (w_4^{3,2},w_5^{3,2},z_0)\ (w_5^{3,2},w_6^{3,2},y_3)\ (w_6^{3,2},w_7^{3,2},y_2)
\end{array}
$$

$(w_1^{3,3},w_2^{3,3},z_1)$ $(w_2^{3,3},w_3^{3,3},z_1)$ $(w_3^{3,3},w_4^{3,3},z_1)$ $(w_4^{3,3},w_5^{3,3},z_1)$ $(w_5^{3,3},w_6^{3,3},y_3)$ $(w_6^{3,3},w_7^{3,3},y_3)$

Let $[v] \stackrel{def}{=} \{0,..,v-1\}$. We define the 1-1 matching M_π between $Trans(C_1)$ and $Trans(C_2)$ as follows:

$$\{x_i - y_{\pi(i)} \Rightarrow X_{\pi(i)}\}_{0\le i<v} \cup \{z_0 - z_0, z_1 - z_1\} \cup \tag{6}$$

$$\{u_k^{l,\pi(l),\pi(l+1)} - w_k^{\pi(l),\pi(l+1)} \Rightarrow W_k^l\}_{1\le k\le 2\log v+3 \text{ and } 0\le l<v-1} \cup \tag{7}$$

$$\{u_k^{0,i,j} - w_{2\log v+4-k}^{i,j}\}_{1\le k\le 2\log v+3 \text{ and } (i,j)\in[v]^2\setminus\{(\pi(l),\pi(l+1))\,|\,0\le l<v-1\}} \tag{8}$$

First we note that this is indeed a 1-1 matching since no variable in $Trans(C_1)$ or $Trans(C_2)$ is used twice in M_π, and all variables in $Trans(C_2)$ are present in it (the clause $Trans(C_2)$ has fewer variables than $Trans(C_1)$).

Parts (7) and (8) determine the matchings between auxiliary variables (those coming from the transformation $Trans$); part (6) matches original variables. As we see next, (7) is designed so that atoms in $lgg_{\pi\cup\{z_0-z_0,z_1-z_1\}}(C_1,C_2)$ are included in the pairing and (8) guarantees that everything else is not included in the pairing.

We carefully study $lgg_{M_\pi}(Trans(C_1), Trans(C_2))$. We observe that M_π matches auxiliary variables $u_*^{*,i,j} - w_*^{i,j}$. Therefore atoms are included in the pairing only if they are produced by $P_{*,i,j,*,*} \in Trans(C_1)$ and $P_{i,j,**} \in Trans(C_2)$ sharing the same i,j. In the case that $i = \pi(l)$ and $j = \pi(l+1)$ for some $l \in \{0,..,v-2\}$, we observe by (7) that the auxiliary variables are matched following their order in the chain $\{u_k^{l,\pi(l),\pi(l+1)} - w_k^{\pi(l),\pi(l+1)} \Rightarrow W_k^l\}_{1\le k\le 2\log v+3}$, and hence clauses $P_{l,\pi(l),\pi(l+1),x_l,x_{l+1}} \in C_1$ and $P_{\pi(l),\pi(l+1),y_{\pi(l)},y_{\pi(l+1)}} \in C_2$ are included in the pairing precisely as

$$P_{\pi(l),\pi(l+1),X_{\pi(l)},X_{\pi(l+1)}} \approx Trans(p(binary(\pi(l)), binary(\pi(l+1)), X_{\pi(l)}, X_{\pi(l+1)}))$$

where the auxiliary variables used in the transformation are $W_1^l,..,W_{2\log v+3}^l$. To see that atoms in the product $P_{*,i,j,*,*} \times P_{i,j,**}$ are not included in the pairing when $(i,j) \in [v]^2 \setminus \{(\pi(l),\pi(l+1)) \,|\, 0 \le l < v-1\}$, it is sufficient to observe that the auxiliary variables are matched in reversed order $\{u_k^{0,i,j} - w_{2\log v+4-k}^{i,j}\}$ (8), so that in order to be included it is required that an atom $p(w_{k+1}^{i,j}, w_k^{i,j}, *)$ exists in $Trans(C_2)$ which is not possible by construction. Therefore:

$$\begin{aligned}
&lgg_{M_\pi}(Trans(C_1), Trans(C_2)) \\
&\quad\approx \bigvee_{0\le l<v-1} P_{\pi(l),\pi(l+1),X_{\pi(l)},X_{\pi(l+1)}} \\
&\quad\approx \bigvee_{0\le l<v-1} Trans(p(binary(\pi(l)), binary(\pi(l+1)), X_{\pi(l)}, X_{\pi(l+1)})) \\
&\quad\approx Trans(\bigvee_{0\le l<v-1} p(binary(\pi(l)), binary(\pi(l+1)), X_{\pi(l)}, X_{\pi(l+1)})) \\
&\quad\approx Trans(lgg_{\pi\cup\{z_0-z_0,z_1-z_1\}}(C_1,C_2)).
\end{aligned}$$

Example 5.4 *Following Example 5.3, the matching $M_{(3201)}$ is as follows. Corresponding to (6):*

$$\{x_0 - y_3 \Rightarrow X_3, x_1 - y_2 \Rightarrow X_2, x_2 - y_0 \Rightarrow X_0, x_3 - y_1 \Rightarrow X_1\} \cup \{z_0 - z_0, z_1 - z_1\}$$

Corresponding to (7):

$$\{u_1^{0,3,2} - w_1^{3,2} \Rightarrow W_1^0, u_2^{0,3,2} - w_2^{3,2} \Rightarrow W_2^0, \ldots, u_7^{0,3,2} - w_7^{3,2} \Rightarrow W_7^0\} \cup$$
$$\{u_1^{1,2,0} - w_1^{2,0} \Rightarrow W_1^1, u_2^{1,2,0} - w_2^{2,0} \Rightarrow W_2^1, \ldots, u_7^{1,2,0} - w_7^{2,0} \Rightarrow W_7^1\} \cup$$
$$\{u_1^{2,0,1} - w_1^{0,1} \Rightarrow W_1^2, u_2^{2,0,1} - w_2^{0,1} \Rightarrow W_2^2, \ldots, u_7^{2,0,1} - w_7^{0,1} \Rightarrow W_7^2\}$$

Corresponding to (8):

$$\{u_1^{0,0,0} - w_7^{0,0}, u_2^{0,0,0} - w_6^{0,0}, \ldots, u_7^{0,0,0} - w_1^{0,0}\} \cup \{u_1^{0,0,2} - w_7^{0,2}, u_2^{0,0,2} - w_6^{0,2}, \ldots, u_7^{0,0,2} - w_1^{0,2}\} \cup$$
$$\{u_1^{0,0,3} - w_7^{0,3}, u_2^{0,0,3} - w_6^{0,3}, \ldots, u_7^{0,0,3} - w_1^{0,3}\} \cup$$
$$\{u_1^{0,1,0} - w_7^{1,0}, u_2^{0,1,0} - w_6^{1,0}, \ldots, u_7^{0,1,0} - w_1^{1,0}\} \cup \{u_1^{0,1,1} - w_7^{1,1}, u_2^{0,1,1} - w_6^{1,1}, \ldots, u_7^{0,1,1} - w_1^{1,1}\} \cup$$
$$\{u_1^{0,1,2} - w_7^{1,2}, u_2^{0,1,2} - w_6^{1,2}, \ldots, u_7^{0,1,2} - w_1^{1,2}\} \cup \{u_1^{0,1,3} - w_7^{1,3}, u_2^{0,1,3} - w_6^{1,3}, \ldots, u_7^{0,1,3} - w_1^{1,3}\} \cup$$
$$\{u_1^{0,2,1} - w_7^{2,1}, u_2^{0,2,1} - w_6^{2,1}, \ldots, u_7^{0,2,1} - w_1^{2,1}\} \cup \{u_1^{0,2,2} - w_7^{2,2}, u_2^{0,2,2} - w_6^{2,2}, \ldots, u_7^{0,2,2} - w_1^{2,2}\} \cup$$
$$\{u_1^{0,2,3} - w_7^{2,3}, u_2^{0,2,3} - w_6^{2,3}, \ldots, u_7^{0,2,3} - w_1^{2,3}\} \cup$$
$$\{u_1^{0,3,0} - w_7^{3,0}, u_2^{0,3,0} - w_6^{3,0}, \ldots, u_7^{0,3,0} - w_1^{3,0}\} \cup \{u_1^{0,3,1} - w_7^{3,1}, u_2^{0,3,1} - w_6^{3,1}, \ldots, u_7^{0,3,1} - w_1^{3,1}\} \cup$$
$$\{u_1^{0,3,3} - w_7^{3,3}, u_2^{0,3,3} - w_6^{3,3}, \ldots, u_7^{0,3,3} - w_1^{3,3}\} \cup$$

Notice that the portion of the matching

$$\{u_1^{0,3,2} - w_1^{3,2} \Rightarrow W_1^0, u_2^{0,3,2} - w_2^{3,2} \Rightarrow W_2^0, \ldots, u_7^{0,3,2} - w_7^{3,2} \Rightarrow W_7^0\}$$

makes sure that the atoms $P_{0,3,2,x_0,x_1}$ in Trans(C_1)

$$(u_1^{0,3,2},u_2^{0,3,2},z_1)\ (u_2^{0,3,2},u_3^{0,3,2},z_1)\ (u_3^{0,3,3},u_4^{0,3,3},z_1)\ (u_4^{0,3,2},u_5^{0,3,2},z_0)\ (u_5^{0,3,2},u_6^{0,3,2},x_0)\ (u_6^{0,3,2},u_7^{0,3,2},x_1)$$

and the atoms $P_{3,2,y_3,y_2}$ in Trans(C_2)

$$(w_1^{3,2},w_2^{3,2},z_1)\ (w_2^{3,2},w_3^{3,2},z_1)\ (w_3^{3,2},w_4^{3,2},z_1)\ (w_4^{3,2},w_5^{3,2},z_0)\ (w_5^{3,2},w_6^{3,2},y_3)\ (w_6^{3,2},w_7^{3,2},y_2)$$

appear in the pairing $lgg_{M_{(3201)}}$(Trans(C_1), Trans(C_2)) as

$$(W_1^0,W_2^0,z_1)\ (W_2^0,W_3^0,z_1)\ (W_3^0,W_4^0,z_1)\ (W_4^0,W_5^0,z_0)\ (W_5^0,W_6^0,X_3)\ (W_6^0,W_7^0,X_2).$$

Finally, $lgg_{M_{(3201)}}$(Trans(C_1), Trans(C_2)) =

$$(W_1^0,W_2^0,z_1)\ (W_2^0,W_3^0,z_1)\ (W_3^0,W_4^0,z_1)\ (W_4^0,W_5^0,z_0)\ (W_5^0,W_6^0,X_3)\ (W_6^0,W_7^0,X_2)$$
$$(W_1^1,W_2^1,z_1)\ (W_2^1,W_3^1,z_0)\ (W_3^1,W_4^1,z_0)\ (W_4^1,W_5^1,z_0)\ (W_5^1,W_6^1,X_2)\ (W_6^1,W_7^1,X_0)$$
$$(W_1^2,W_2^2,z_0)\ (W_2^2,W_3^2,z_0)\ (W_3^2,W_4^2,z_0)\ (W_4^2,W_5^2,z_1)\ (W_5^2,W_6^2,X_0)\ (W_6^2,W_7^2,X_1)$$

Recall $lgg_{\pi\cup\{z_0-z_0,z_1-z_1\}}(C_1,C_2)$ *is*

$$(z_1,z_1,z_1,z_0,X_3,X_2)\ (z_1,z_0,z_0,z_0,X_2,X_0)\ (z_0,z_0,z_0,z_1,X_0,X_1)$$

and hence $Trans(lgg_{\pi\cup\{z_0-z_0,z_1-z_1\}}(C_1,C_2))$ *is*

$$\begin{array}{l}
(Y_1^1,Y_2^1,z_1)\ (Y_2^1,Y_3^1,z_1)\ (Y_3^1,Y_4^1,z_1)\ (Y_4^1,Y_5^1,z_0)\ (Y_5^1,Y_6^1,X_3)\ (Y_6^1,Y_7^1,X_2)\\
(Y_1^2,Y_2^2,z_1)\ (Y_2^2,Y_3^2,z_1)\ (Y_3^2,Y_4^2,z_1)\ (Y_4^2,Y_5^2,z_0)\ (Y_5^2,Y_6^2,X_2)\ (Y_6^2,Y_7^2,X_0)\\
(Y_1^3,Y_2^3,z_1)\ (Y_2^3,Y_3^3,z_1)\ (Y_3^3,Y_4^3,z_1)\ (Y_4^3,Y_5^3,z_0)\ (Y_5^3,Y_6^3,X_0)\ (Y_6^3,Y_7^3,X_1)
\end{array}$$

and indeed one can check that

$$lgg_{M_{(3201)}}(Trans(C_1),Trans(C_2)) \approx Trans(lgg_{(3201)\cup\{z_0-z_0,z_1-z_1\}}(C_1,C_2))$$

via the variable renaming $\{Y_k^1 \leftrightarrow W_k^0, Y_k^2 \leftrightarrow W_k^1, Y_k^3 \leftrightarrow W_k^2 \mid 1 \le k \le 7\}$.

Theorem 5.3 *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* $O(v^3 \log v)$ *variables,* $O(v^3 \log v)$ *literals can be* $\Omega(v!)$. ■

Renaming parameters to use $v^{1/4}$ original variables in the theorem we get:

Corollary 5.4 *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* $\Omega(2^{v/4})$. ■

5.4 Implications for Learnability

The Algorithms in [14, 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 $v_1, \ldots, v_{N_1}$, and C_2 uses variables $w_1, \ldots, w_{N_2}$. Then we use $q()$ and $r()$ to define chains of variables touching all variables in C_1, C_2: $Q = \wedge_{1 \leq l < N_1} q(v_l, v_{l+1})$ and $R = \wedge_{1 \leq l < N_2} r(w_l, w_{l+1})$. 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 $T = D_1 \wedge D_2$. The following 3 lemmas give useful properties of T and its clauses.

Lemma 5.5 *$D_1 \not\preceq D_2$ and hence $D_1 \not\models D_2$. $D_2 \not\preceq D_1$ and hence $D_2 \not\models D_1$.*

Proof: To see that $D_1 \not\preceq D_2$ and $D_2 \not\preceq D_1$, it suffices to notice that in D_1 there are atoms containing the predicate symbol q which is not present in D_2, and in D_2 there are atoms containing the predicate symbol r which is not present in D_1. $D_1 \not\models D_2$ and $D_2 \not\models D_1$ follow from the fact that when considering clauses that are not self-resolving and where chaining is not possible, logical implication coincides with subsumption [11]. ■

Lemma 5.6 *Let C be any clause. Let $T = D_1 \wedge D_2$ as defined above. If $T \models C$ then it is the case that either $D_1 \preceq C$ or $D_2 \preceq C$.*

Proof: Since the clauses in T are not self-resolving and D_1 and D_2 cannot be resolved together, implication reduces to subsumption [11]. ■

Lemma 5.7 *Let $T = D_1 \wedge D_2$ as defined above. If D is a result of dropping any object from D_1 or D_2 then $T \not\models D$.*

Proof: By Lemma 5.6, $T \models D$ iff $D_1 \preceq D$ or $D_2 \preceq D$. Assume D is a result of dropping an object from D_1 (the other case is similar). Then since Q, R use different predicate symbols it is clear that $D_2 \not\preceq D$. To see that $D_1 \not\preceq D$ notice that D_1 includes a $q()$ chain of length N_1 including all variables in D_1. Consider any substitution θ mapping variables in D_1 to variables in D and assume that v_1 is mapped to w_k for some k. The only way for subsumption to work is to map v_2 to w_{k+1} and so on. However, since D contains strictly fewer variables than D_1 then for some i it must be the case that $q(w_{k+i}, w_{k+i+1})$ is not in D. Therefore it cannot be the case that $D_1\theta \subseteq D$. ■

Lemma 5.7 establishes condition (1). It is easy to see that the clauses have exactly the same pairings as in the previous section since the atoms by q and r are dropped in every pairing and the chains do not introduce new variables. This implies that condition (2) holds. Finally (3) holds since q and r atoms are dropped in every pairing so subsumption is not possible. We therefore get:

Theorem 5.8 *The algorithm of [3] can make $\Omega(2^{v/4})$ queries on some learning problems.*

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] G. Gottlob. Subsumption and implication. *Information Processing Letters*, 24(2):109–111, 1987.

[12] 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.

[13] L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan, and D. Wilkins. How many queries are needed to learn? *Journal of the ACM*, 43(5):840–862, September 1996.

[14] R. Khardon. Learning function free Horn expressions. *Machine Learning*, 37:241–275, 1999.

[15] J. W. Lloyd. *Foundations of logic programming; (2nd extended ed.)*. Springer-Verlag New York, Inc., 1987.

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

[17] G. D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[18] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[19] 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.

[20] 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.

[21] 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.

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