TUFTS-CS Technical Report 2004-5

June 2004

Exact Learning of First-Order Expressions from Queries

by

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

EXACT LEARNING OF FIRST-ORDER EXPRESSIONS FROM QUERIES

A thesis

submitted by

Marta Arias

In partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in

Computer Science

TUFTS UNIVERSITY

May, 2004

Adviser: Roni Khardon

Abstract

This thesis studies the complexity of learning logical expressions in the model of Exact Learning from Membership and Equivalence Queries. The focus is on Horn expressions in first order logic but results for propositional logic are also derived. The thesis includes several contributions towards characterizing the complexity of learning in these contexts.

First, a new algorithm for learning first order Horn expressions is presented and proved correct. The algorithm improves on previous work in two ways. It can learn a larger class of expressions than previously known, and its query and time complexity improve on previous algorithms. In particular the algorithm can learn both the class of range-restricted expressions, and the class of constrained expressions, which were previously considered in the literature.

Second, the thesis includes several lower bound results and techniques studying the optimal complexity of these learning problems, thus trying to identify whether it is possible to improve over the complexity of our algorithm.

We study the VC Dimension of Horn expressions, a tool that gives lower bounds on query complexity in our model. Our results characterize exactly the VC Dimension of Horn expressions, providing the best lower bound possible using this technique. This technique leaves a gap to the upper bound provided by our algorithm. Our analysis also highlights problematic aspects of measuring learning complexity in first order logic that have been ignored in previous work.

We study the Certificate Size, a tool that characterizes the query complexity of learning in our model. Our results give certificate constructions for several classes

of important propositional expressions, including Horn CNF expressions. We also show that any certificate for a slight generalization of Horn CNF expressions, the class of renamable Horn CNF expressions, is of exponential size, thus showing that this class is not efficiently learnable.

Finally, we study the lattice structure induced by generality relations in first order logic expressions, and derive some conclusions for learning complexity in more restricted scenarios.

Acknowledgments

I am grateful to my advisor, Roni Khardon, who has always supported and guided me. Without his help I would have never been able to write this thesis. He has been a wonderful mentor, teacher and friend. I also want to thank the other members of my committee who made very helpful comments about this work: Dana Angluin, Anselm Blumer, Christoph Börgers and Lenore Cowen. In particular, I want to thank Lenore Cowen for encouraging me to continue my academic career. I am grateful to all the members of the Computer Science Department at Tufts who have created a wonderful environment to work in.

I would like to thank Rocco Servedio, José Luis Balcázar, Rajmohan Rajaraman, Kofi Laing and Orjeta Taka for stimulating discussions and their collaboration.

I am grateful for the financial support provided by EPSRC Grant GR/M21409 while at Edinburgh, and by NSF Grant IIS-0099446 while at Tufts.

I am grateful to all the wonderful friends I have met along the way: Milena Maule, Patrick McAvoy, Alastair Borrowman, Siddharth Sarin, Mercedes Balcells, Jeff Comfort, Ana Rodríguez, Marta Cortés, Thomas Würz, Toshimi Yoshida, Janira Arocho, Santi Pathak, Chun Yu, Marcelo Coca, Eduardo Calvillo, Orjeta Taka, Arthur Brady, Peter Waltman, Andrew McDonnell, Eynat Rafalin, Julene Mazpule, Blanca San Miguel, Ana Sala, William Oliver, Ashley Englander, and many others who I hope can forgive me for not writing their names.

Finally, I would like to thank my family for all the support and love they have always given me.

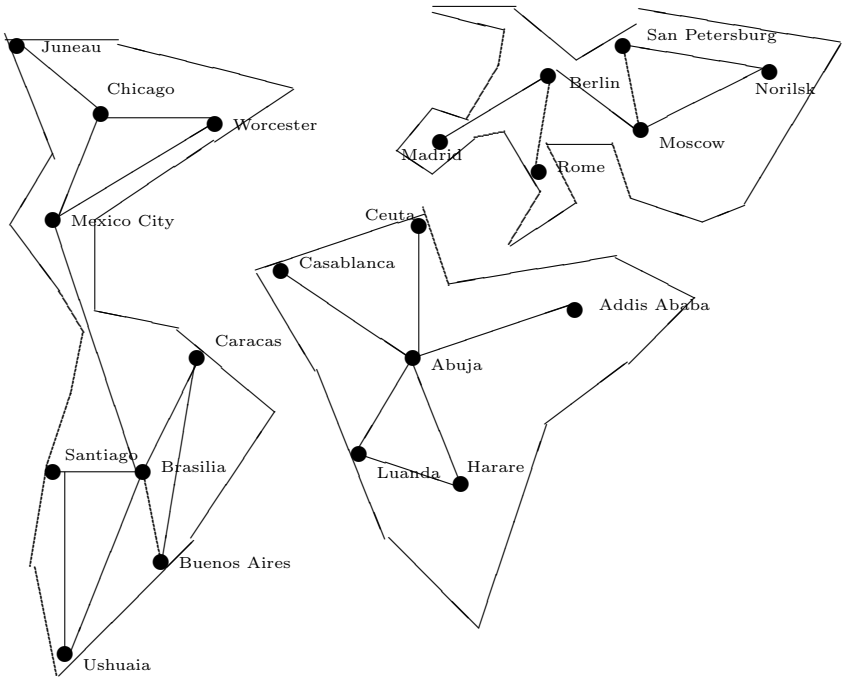
Contents

EXACT LEARNING OF FIRST-ORDER EXPRESSIONS FROM QUERIES

Chapter 1

Introduction

This thesis is concerned with the problem of learning concepts expressed in first order logic. First order logic is a highly expressive language that allows us to describe complicated phenomena concisely. As an example, take the following imaginary map of cities:



To express the concept "*two cities are connected by a major highway*" we just need the following first order rule: "for all cities x, y and z, if x is connected to y

and y is connected to z, then x is connected to z" which we formally write in the language of first order logic as:

$$\forall x \forall y \forall z \; connected(x, y) \wedge connected(y, z) \rightarrow connected(x, z) \tag{1.1}$$

From such a rule we can extract all the pairs of cities that are connected, just by applying it repeatedly and assuming that initially the connected cities are those adjacent in our map. If we were to use a propositional formalism to describe the same concept we would have to explicitly list the cities that are connected in our map, partially represented by the following table that contains $21^2 = 441$ entries for a map with 21 cities:

City 1	*City 2*	*Connected?*	*City 1*	*City 2*	*Connected?*
Casablanca	Ceuta	YES	Caracas	Casablanca	NO
Casablanca	Abuja	YES	Casablanca	Harare	YES
Harare	Abuja	YES	Harare	Caracas	NO
⋮	⋮	⋮	⋮	⋮	⋮

Notice also that if we were to change the map, the propositional description would have to change in order to reflect the changes made. However, the first order description of the concept remains unchanged since it refers to the transitive nature of connectedness and hence it applies to every map imaginable.

In this work we study the complexity of learning first order Horn expressions which are essentially sets of rules such as 1.1. We adopt a supervised learning scenario that assumes that there is a teacher (or *oracle*) available that answers questions (or *queries*) posed by the learning algorithms. We assume that there is a set of first order rules known only to the teacher; the task of the learning algorithm is to discover these rules (or a set of equivalent rules) by asking questions to the teacher. It is important to distinguish this learning scenario from the more passive one where algorithms are just presented with a series of labeled examples and no active questions are allowed.

Interest in learning concepts expressed in logic is not new. In fact, during the last two decades the machine learning community has produced an impressive list

of results related to learning different types of logic formalisms, both theoretical and empirical. For example, learning algorithms for decision trees or rule systems have been developed and applied to numerous real-world problems very successfully (Mitchell, 1997).

Inductive Logic Programming

In the context of learning in first order logic, Plotkin (1970) introduces in his pioneering work an algorithm for computing the least general generalization of a pair of clauses (w.r.t. a generalization relation known as subsumption — see Section 2.3 for details). This algorithm was later incorporated into a whole theory of inductive learning in first order clausal logic (Plotkin, 1971). The notion of least general generalization and subsumption is still central to first order learning, and in particular, central to our learning algorithm in Chapter 5. Another example of early work in first order logic learning is that of Shapiro (1983), where he formalizes the *model inference problem* that is the problem of inferring a first order theory that is capable of explaining some observed facts. Based on his theory, he develops the *model inference system* and applies it to the problem of debugging logic programs. Other early examples of theoretical studies of the complexity of learning in first order logic are (Valiant, 1985; Haussler, 1989). Learning first order logic is currently studied under the name of *Inductive Logic Programming* (ILP). Work in ILP ranges from applications, the development of systems and algorithms, to theoretical studies. While the work presented in this thesis is purely theoretical, it is of interest to follow the evolution of machine learning systems developed within the ILP community.

ILP systems can be viewed as algorithms that perform a search guided by examples or queries (or both) over the lattice formed by first order clauses and their subsumption relation — see Section 2.3 for details. Early ILP systems such as CIGOL (Muggleton and Buntine, 1988) or GOLEM (Muggleton and Feng, 1992) perform a bottom-up search, i.e. they produce hypotheses that are increasingly more general, starting with the most specific hypothesis. All these systems suffer from very high

computational costs, and top-down systems were developed to improve on their efficiency. Systems such as FOIL (Quinlan, 1990), PROGOL (Muggleton, 1995) and ICL (De Raedt and Van Laer, 1995) use a greedy covering method where the system adds one clause at a time to the hypothesis and each clause is constructed by a general to specific refinement search. LogAn-H (Khardon, 2000), based on the algorithm in (Khardon, 1999b), is the first bottom-up system introduced after some time.

While most of ILP systems are based on examples, some of the early systems learn from queries: MIS (Shapiro, 1983), MARVIN (Sammut and Banerji, 1986), CIGOL (Muggleton and Buntine, 1988), CLINT (De Raedt and Bruynooghe, 1992) and also the more recent LogAn-H (Khardon, 2000). Some of these systems require the presence of an expert to answer the questions posed by the system. This is for example the case of MIS (Shapiro, 1983) where the system is integrated in a programming development and debugging environment. In the case of MIS it seems natural that the expert (the programmer) is available during the process. However, this is quite rare and query-based systems usually simulate the answers to the queries using examples. This is easy in the case of equivalence queries, which ask whether a given hypothesis is correct or not. Here the hypothesis can be tested against a set of examples and if a discrepancy is found then the answer to the query is `No` whereas if no discrepancy is found the answer should be `Yes`. It is well known that if the set of examples is large enough, then good guarantees about the accuracy of the hypothesis can be obtained (Angluin, 1988). Membership queries are usually harder to simulate and a more ad-hoc solution has to be found for every system. This is in fact what LogAn-H (Khardon, 2000) does in its "batch" mode, so that no interaction from the user is required and the system just runs using a database of labeled examples. Query-based learning algorithms can also be used within larger systems that somehow are able to experiment with the hypotheses provided by the systems. This is the case of the experiments of Reddy and Tadepalli (1999) in the context of planning. In this system, they can test the hypotheses generated by their

learning algorithm (Reddy and Tadepalli, 1997) by simulating the planning process using the hypothesis output by the learning algorithm. The same idea applies to the work of Bryant and Muggleton (2000) and also Muggleton et al. (1999), whose aim is to automate the scientific discovery process by having a machine learning algorithm proposing hypotheses and a robot testing the hypotheses by performing some experiments.

Complexity of learning in first order logic

Unfortunately, Cohen (1995) shows that efficient learning algorithms do not exist even for very simple classes of first order concepts. These negative results apply to the PAC learning model (Valiant, 1984), where examples are drawn according to some unknown distribution and algorithms have no control over which examples they are allowed to see. To overcome this, we relax the problem by allowing algorithms to actively select examples. More precisely, we consider the stronger model of exact learning from membership and equivalence queries (Angluin, 1988). Informally, in an equivalence query, the learning algorithm suggests a hypothesis and an oracle answers `Yes` or `No` depending on whether the hypothesis is (logically) equivalent to the target concept or not. In a membership query, the learning algorithm presents an example and an oracle returns `Yes` if it is a member of the target concept, otherwise it returns `No`. The model of exact learning from membership and equivalence queries has been studied extensively, mostly in the context of learning in propositional logic. Indeed, some problems that are provably hard in the PAC model[1] (or still open) become tractable when queries are allowed. Examples of this are propositional Horn expressions (Angluin, Frazier, and Pitt, 1992; Frazier and Pitt, 1993), read once formulas (Angluin, Hellerstein, and Karpinski, 1993), k-term DNF formulas for fixed k (Angluin, 1987a), regular sets (Angluin, 1987b) and monotone DNF formulas (Angluin, 1988; Valiant, 1984), among others.

[1]Hardness results in the PAC model are commonly proved assuming plausible conditions such as $P \neq NP$ or $RP \neq NP$ or assuming that certain cryptographic problems are hard.

As we have mentioned, early studies of learning in relational domains using queries can be found in Shapiro (1983), Valiant (1985) and Haussler (1989). Recently, algorithms have been developed in the model of exact learning from queries capable of identifying expressive subsets of first order Horn expressions. Some of these algorithms use more powerful types of queries that partially reveal some of the syntactic structure of the target concepts such as *subsumption queries* (Arimura, 1997; Rao and Sattar, 1998) or *derivation order queries* (Reddy and Tadepalli, 1998). Other algorithms, including our own in Chapter 5 (Reddy and Tadepalli, 1997; Khardon, 1999a; Khardon, 1999b; Arias and Khardon, 2002) use the standard membership and equivalence queries only. All of these algorithms resemble the propositional learning algorithms of Angluin, Frazier, and Pitt (1992) and Frazier and Pitt (1993). In fact, they can all be viewed as a generalization of these algorithms to the first order setting. Naturally, the operations of the propositional algorithm need to be lifted to first order logic. This is done by using variants of the *least general generalization* or *lgg* of Plotkin (1970) or by using direct products of first order interpretations and other appropriate operations.

Results

One of the main contributions of this thesis is in presenting a learning algorithm for an important class of first order Horn expressions (Chapter 5). This result improves on earlier work by learning a larger subset of first order Horn expressions with provably fewer queries. The learning algorithm uses equivalence queries to test whether its incrementally constructed hypotheses are equivalent to the target concept, and uses a variant of the least general generalization (Plotkin, 1970) and membership queries to update incorrect hypotheses.

To quantify the complexity of our learning algorithm, we use parameters that are based on the syntactic components used to describe the target concept. Important parameters are the number of clauses, the maximum number of variables in a clause v (in our example clause 1.1, $v = 3$ due to variables x, y and z), the maximum

number of terms in a clause t (in our example clause 1.1, $t = 3$ also since the variables are the only terms), and the maximum number of literals in a clause l (in our example clause 1.1, $l = 3$ due to two atoms in the antecedent and one in the consequent). The use of such syntax-based parameters is common in ILP, however, there has never been a theoretical justification for this. Chapter 4 introduces a series of parameters that quantify the complexity of first order expressions. It also includes two fundamentally different ways of computing the *size* of an expression: *TreeSize* and *DAGSize*. *TreeSize* is considered the standard notion of size for first order expression and counts essentially the number of symbols needed to write down a first order expression in its usual string form. *DAGSize* is based on a more efficient encoding of the expressions that avoids repetitions of multiple occurrences of identical terms. Chapter 4 provides a framework that characterizes under which circumstances it is justified to use one set of parameters or another. It relates *DAGSize* to the three parameters c, l, t and shows that no combination of parameters can relate to *TreeSize*. From this we conclude that it is sufficient to take into account the three parameters c, l and t if one wants an algorithm that is efficient w.r.t. *DAGSize*.

The complexity of our algorithm in Chapter 5 is exponential in the number of variables. This contrasts with the algorithms of (Arimura, 1997; Reddy and Tadepalli, 1998; Rao and Sattar, 1998) whose complexity is only polynomial in this crucial parameter. However, as we mentioned earlier, these algorithms use very powerful queries. It is thus interesting to investigate whether this exponential dependence is necessary, or in other words, whether one can find better algorithms if only membership and equivalence queries are available.

Chapter 6 takes a first step in this direction by studying the VC Dimension of first order Horn expressions. The VC Dimension of a class is known to give a lower bound for the number of queries needed to learn the class when using membership and equivalence queries (Maass and Turán, 1992). In Chapter 6 we show that the VC Dimension of first order Horn expressions is not exponential in the number of

variables (it is polynomial in all the relevant parameters) and hence it leaves a gap to the exponential upper bound provided by the learning algorithm. The remainder of this thesis is concerned with closing this gap.

After the introduction of the model of exact learning from queries (Angluin, 1988) there has been great effort put into characterizing learnability when certain types of queries are available by means of combinatorial quantitative characterizations of the concept classes to be learnt. A summary of these combinatorial characterizations can be found in (Angluin, 2001). Particularly relevant to this work is the notion of *certificate size*, which is directly related to the number of queries needed to learn from membership and equivalence queries (Hellerstein et al., 1996; Hegedus, 1995). Chapter 7 studies the certificate size of various classes of propositional expressions, including propositional Horn expressions. Constructions of certificates of polynomial size are given for unate CNF/DNF and Horn CNF; these can be viewed as alternative proofs of their learnability. Some matching lower bounds for certificate size are also given. Chapter 7 shows also that renamable Horn CNF, a slight generalization of propositional Horn CNF, has certificates of exponential size. This implies that there is no polynomial algorithm that learns this slightly more general class and solves an open question of Feigelson (1998).

Finally, Chapter 8 studies some properties of the subsumption lattice over first order clauses. We first show that subsumption chains of exponential length exist; this fact is then used to show the impossibility of efficiently learning first order Horn clauses in the restricted model where only membership queries are available. Then we show that a pair of clauses can have an exponential number of pairings (a *pairing* is a variant of the *lgg* used by our learning algorithm to generalize clauses). This means that there are cases in which the learning algorithm in Chapter 5 must make an exponential number of queries, thus showing that our analysis is tight.

In summary, the thesis provides a new algorithm and complexity upper and lower bounds for the problem of learning first order Horn expressions. Chapter 9 includes further discussion of the results and directions for further work.

Chapter 2

Logic Review

In this chapter we review the standard definitions and results of logic and that are used in this thesis. We do not attempt to give a comprehensive review of logic; readers unfamiliar with mathematical logic can refer to standard textbooks e.g. (Lloyd, 1987; Chang and Keisler, 1990).

2.1 Propositional logic

2.1.1 Syntax

Expressions in propositional logic (also called *formulas*) are built using a non-empty, finite set of propositional variables $V = \{v_1, .., v_n\}$ with $n \in \mathbb{N}^+$, the logical connectives '$\neg$' (negation), '$\wedge$' (conjunction), '$\vee$' (disjunction), '$\rightarrow$' (implication), and two punctuation symbols '(' and ')' used to resolve ambiguities with the logical connectives. An example of a well-formed formula is '$(v_1 \wedge \neg v_3) \vee (v_2 \wedge v_3 \rightarrow v_4)$'.

Of particular importance are formulas that are in *Disjunctive Normal Form* or DNF which is a disjunction of conjunctions, and its dual the *Conjunctive Normal Form* or CNF which is a conjunction of disjunctions. The formula above has as DNF representation '$(v_1 \wedge \neg v_3) \vee \neg v_2 \vee \neg v_3 \vee v_4$' and as CNF representation '$(v_1 \vee \neg v_2 \vee \neg v_3 \vee v_4) \wedge (\neg v_3 \vee \neg v_2 \vee \neg v_3 \vee v_4)$'.

In propositional logic, a conjunction of literals such as '$v_1 \wedge \neg v_3$' is called a *term*. A disjunction such as '$\neg v_3 \vee \neg v_2 \vee \neg v_3 \vee v_4$' is called a *clause*. A clause which has at most one positive variable is called a *Horn clause*. Examples of Horn clauses are '$\neg v_3 \vee \neg v_2 \vee \neg v_3 \vee v_4$' and also '$\neg v_3 \vee \neg v_2 \vee \neg v_3$'. A Horn clause is usually denoted by an implication '$A \rightarrow a$', where A is a conjunction of positive variables and a is a positive variable. A is commonly referred to as the *antecedent* of the clause and a as its *consequent*. In the Horn clause '$A \rightarrow a$', the variables in A are negative, and a is the (at most one) positive variable. Both A and a can be empty. As an example, the Horn clause '$\neg v_3 \vee \neg v_2 \vee v_4$' is written as '$v_3 \wedge v_2 \rightarrow v_4$', and the Horn clause '$\neg v_3$' is written as '$v_3 \rightarrow$'. A Horn CNF formula is a conjunction of Horn clauses.

We note that there are many conventions concerning the priorities of the logical connectives and the usage of the parentheses that we do not describe here.

We also note that in this section we have written formulas in quotes ' ' to distinguish them from regular text. From now on we stop doing this, and hope that notation and context are enough to make this distinction.

2.1.2 Semantics

An *assignment* assigns a truth value (we use '0' and '1') to each propositional variable in $V = \{v_1, .., v_n\}$. It is typically denoted by a string in $\{0,1\}^n$. For example, assuming $n = 5$, the assignment 00110 assigns v_1 to 0, v_2 to 0, v_3 to 1, v_4 to 1, and v_5 to 0.

Given an assignment x and a formula f, we can *evaluate* the truth or falsity of f under x in the following way: first substitute every occurrence of a variable in f by its corresponding truth value given by x, and then recursively apply the rules dictated by the classical truth tables of the logical connectives to obtain its final value. We say that an assignment x satisfies a formula f, noting this by $x \models f$, if the formula f evaluates to 1 under x. Otherwise, we say that x falsifies the formula f and denote this $x \not\models f$.

Given two formulas f_1 and f_2, we say that f_1 logically implies f_2 and denote this $f_1 \models f_2$ iff for every assignment $x \in \{0,1\}^n$ if $x \models f_1$ then $x \models f_2$. Two formulas f_1, f_2 are logically equivalent (denoted $f_1 \equiv f_2$) iff $f_1 \models f_2$ and $f_2 \models f_1$.

A boolean function $g : \{0,1\}^n \to \{0,1\}$ assigns to every assignment in $\{0,1\}^n$ a value from $\{0,1\}$. Notice that each propositional formula f *represents* a boolean function g in the natural way: for any assignment $x \in \{0,1\}^n$: $g(x) = 1$ if $f \models x$ and $g(x) = 0$ if $f \not\models x$. Different formulas can represent the same boolean function: e.g. the formulas '$(v_1 \wedge \neg v_3) \vee \neg v_2 \vee \neg v_3 \vee v_4$' and '$(v_1 \vee \neg v_2 \vee \neg v_3 \vee v_4) \wedge (\neg v_3 \vee \neg v_2 \vee \neg v_3 \vee v_4)$' represent the same boolean function. Clearly, two formulas representing the same boolean function must be equivalent. We sometimes abuse our notation and identify formulas with their represented boolean functions. I should be clear from the context which one we refer to in each case.

2.2 First order logic

2.2.1 Syntax

A signature $\mathcal{S}$ consists of a set P of predicate symbols (with associated arities) and a set F of function symbols (with associated arities). Syntactically, there is not much difference between function and predicate symbols, with the exception that predicate symbols cannot be nested; the main distinction between them is given by their semantics (see Section 2.2.2). Given a signature $\mathcal{S}$ and a non-empty set of variables V we construct *first order terms*[1] as follows:

- a variable in V is a first order term
- if $f \in F$ is a function symbol of arity a (denoted by f/a), and $t_1, .., t_a$ are first order terms, then $f(t_1, .., t_a)$ is also a first order term; we call these terms *functional terms*

[1] Terms in propositional logic are entirely different from terms in first order logic!

Typically, we use $x, y, z, ..$ to denote variables, $f, g, h, ..$ to denote function symbols. Constants (special function symbols that have arity 0) are denoted by $a, b, c, ..$ and $1, 2, 3, ..$

If $p \in P$ is a predicate symbol of arity a (denoted by p/a) and $t_1, \ldots, t_a$ are first order terms, then $p(t_1, \ldots, t_a)$ is an *atom*. We also consider a special type of atom: the *inequality* $(t_1 \neq t_2)$, where t_1, t_2 are first order terms.

Atoms can be combined using the logical connectives '$\neg$' (negation), '$\wedge$' (conjunction), '$\vee$' (disjunction), '$\rightarrow$' (implication), and the two punctuation symbols '(' and ') of propositional logic into first order formulas (equivalently, we refer to first order formulas by first order expressions). Additionally, first order logic has the *quantifiers* for all '$\forall$' and exists '$\exists$' which allow to quantify variables universally or existentially to form formulas of the sort '$\forall v\ \phi$' or '$\exists v\ \phi$', where v is a variable and ϕ is an arbitrary formula. A *literal* is an atom $p(t_1, \ldots, t_a)$ or its negation $\neg p(t_1, \ldots, t_a)$.

We have seen how to build complex formulas or expressions given a set of variables, a signature $\mathcal{S}$, the logical connectives and the quantifiers. The set of first order expressions built from $\mathcal{S}$ is denoted by $\mathcal{FO}_{\mathcal{S}}$. When we want to make explicit that a formula is in $\mathcal{FO}_{\mathcal{S}}$, we refer to it as a $\mathcal{S}$-formula or $\mathcal{S}$-expression.

Notice that if a signature $\mathcal{S}$ contains predicate symbols of arity 0 only or no function symbols and variables are allowed, then $\mathcal{S}$-expressions are propositional formulas. Hence, propositional logic is a special case of first order logic.

In this work we only consider formulas that are in prenex normal form (the expressions are '$Q_1 v_1 \ldots Q_n v_n\ \phi$' where Q_i are quantifiers and the formula ϕ is quantifier-free). Moreover, we consider *universally quantified* expressions which are expressions that are in prenex normal form and do not contain existential quantifiers.

Given a first order expression $E \in \mathcal{FO}_{\mathcal{S}}$, we define the following sets

- $Vars(E)$ is the set of variables appearing in E
- $Terms(E)$ is the set of first order terms appearing in E, including subterms.

- $Atoms_P(E)$ is the set of atoms that can be built from the terms in $Terms(E)$ and predicate symbols in P.

Example 1 Suppose $P = \{p/2, q/1\}$ and r is a predicate of arity 1.

- $Vars(f(x, g(y))) = \{x, y\}$
- $Terms(f(x, g(a))) = \{x, a, g(a), f(x, g(a))\}$
- $Atoms_P(r(f(1))) = \{p(1,1), p(1, f(1)), p(f(1), 1), p(f(1), f(1)), q(1), q(f(1))\}$

The definition of $Vars$, $Terms$ and $Atoms_P$ can be extended to sets of expressions by taking the union of the result of each individual expression. For example, $Terms(\{E_1, E_2, E_3\}) = Terms(E_1) \cup Terms(E_2) \cup Terms(E_3)$.

A first order clause is a universally quantified disjunction of literals. For example the expression '$\forall x \forall y \forall z\ \neg add(x, y, z) \vee add(y, x, z)$' is a clause. Moreover, it is first order Horn since it contains at most one positive literal. Since all variables are universally quantified, we do not need to write the quantifiers and usually write clauses utilizing the '$\rightarrow$' notation as '$add(x, y, z) \rightarrow add(y, x, z)$'. We also use set notation to denote a clause. In this case, the clause above can be denoted by $\{\neg add(x, y, z), add(y, x, z)\}$. Finally, a first order Horn expression is a conjunction of universally quantified first order Horn clauses.

Definition 1 A first order Horn clause $s \rightarrow b$ is *range restricted* if $Terms(b) \subseteq Terms(s)$. A first order Horn clause $s \rightarrow b$ is *constrained* if $Terms(s) \subseteq Terms(b)$.

For example, the clause $p(x) \rightarrow p(f(x))$ is constrained but not range restricted. On the other hand, the clause $\neg add(x, 0, succ(x))$ is range restricted but not constrained.

Definition 2 A Horn clause is *non-trivial* if it is not a tautology.

Definition 3 Let $ineq(s)$ be the set of all possible inequalities between first order terms appearing in s. A first order Horn clause $s \rightarrow b$ is *fully inequated* if its

antecedent contains all the possible inequalities between pairs of first order terms in it, i.e., if $ineq(s \cup \{b\}) \subseteq s$.

As an example, let s be the set $\{p(x,y), q(f(y))\}$ containing the first order terms $\{x, y, f(y)\}$. Then $ineq(s) = \{x \neq y, x \neq f(y), y \neq f(y)\}$ also written as $(x \neq y \neq f(y))$ for short.

Definition 4 A *meta-clause* is a set of Horn clauses that share the same antecedent. A meta-clause is denoted by $[s, c]$ where s and c are sets of atoms. Formally, $[s, c] \stackrel{def}{=} \bigwedge_{b \in c}(s \rightarrow b)$.

Meta-clauses provide a compact way to represent sets of clauses with the same antecedent and are extensively used in Chapter 5 of this thesis.

Definition 5 A *range restricted first order Horn expression* is a conjunction of range restricted first order Horn clauses. Similarly, *constrained Horn expression* is a conjunction of constrained first order Horn clauses. Finally, a *fully inequated first order Horn expression* is a conjunction of fully inequated first order Horn clauses.

2.2.2 Semantics

Given a signature $\mathcal{S}$, an $\mathcal{S}$-interpretation (also called $\mathcal{S}$-model or $\mathcal{S}$-structure) is the first order analogue of the assignment. The elements of an $\mathcal{S}$-interpretation I are:

- a countable set D called domain whose elements are referred to as objects
- for each $f/a \in F$, I defines a *function mapping* $f^I : D^a \rightarrow D$. Notice that function mappings assign a domain object to each constant: $a^I = o$. The function mappings guide how first order terms are evaluated to domain objects
- for each $p/a \in P$, I includes a subset of $\{P(o_1, .., o_a) \mid o_i \in D, 1 \leq i \leq a\}$. This is called the *extension* of predicate p and it lists which of the instances of the predicate P are true in I

Now we briefly explain how first order $\mathcal{S}$-formulas are evaluated given an $\mathcal{S}$-structure. Given an $\mathcal{S}$-structure with domain D and a first order expression E, a *variable assignment* (w.r.t. $\mathcal{S}$) is an assignment to each variable in E of an object in D.

Given an $\mathcal{S}$-structure with domain D, a first order expression E and a variable assignment V w.r.t. $\mathcal{S}$, a *term assignment* (w.r.t. $\mathcal{S}$ and V) is defined as:

- Each variable in E is given its assignment according to V.
- Each constant is E given its assignment according to $\mathcal{S}$.
- If $t'_1, \ldots, t'_a$ are the term assignments of $t_1, \ldots, t_a$ and f' is the assignment of the a-ary function symbol f, then $f'(t'_1, \ldots, t'_a) \in D$ is the term assignment of $f(t_1, \ldots, t_a)$.

Let I be an $\mathcal{S}$-interpretation with domain D, let V a variable assignment, and E a first order formula. Then the *truth value* of E can be given as follows:

- If E is an atom $p(t_1, \ldots, t_a)$, then the truth value is 1 iff $p(t'_1, \ldots, t'_a)$ is in I's extension for p, where $t'_1, \ldots, t'_a$ are the term assignments for $t_1, \ldots, t_a$ w.r.t. I and V.
- If E is an inequality $t_1 \neq t_2$, then its truth value is 1 iff the term assignments for t_1, t_2 are the same object t' in D.
- If E is of the form $\neg E_1$, $E_1 \wedge E_2$, $E_1 \vee E_2$ or $E_1 \rightarrow E_2$, then the truth value is given by the usual truth table for $\neg, \wedge, \vee$ and $\rightarrow$.
- If E is of the form $\exists x\ E'$, then the truth value is 1 iff there exists an object $d \in D$ such that E' has truth value 1 w.r.t. I and $V \cup \{x \mapsto d\}$.
- If E is of the form $\forall x\ E'$, then the truth value of the formula is 1 iff for all $d \in D$, E' has truth value 1 w.r.t. I and $V \cup \{x \mapsto d\}$.

If an interpretation I makes an expression T evaluate to 1, then we say that I satisfies T and denote this by $I \models T$. In this case, we also say that I is a model of T. If T evaluates to 0 under I, then we say that I falsifies T and denote this by $I \not\models T$. A first order expression T_1 *entails* or logically implies another expression T_2 (denoted $T_1 \models T_2$) if every model of T_1 is also a model of T_2. Two expressions T_1, T_2 are *logically equivalent* (denoted $T_1 \equiv T_2$) iff $T_1 \models T_2$ and $T_2 \models T_1$.

2.2.3 Deduction

A substitution is a mapping from variables into first order terms. We denote substitutions as sets of ordered pairs $\{x_1 \mapsto t_1, .., x_n \mapsto t_n\}$ where x_i are variables and t_i are first order terms for all $i = 1..n$. We usually refer to substitutions by the Greek letter θ and variations of it. Substitutions can be applied to first order terms, atoms, sets of atoms, and any first order expression in general. The effect of applying a substitution $\theta = \{x_1 \mapsto t_1, .., x_n \mapsto t_n\}$ to a first order expression E, denoted as $E \cdot \theta$, is to (simultaneously) replace the free variables x_i that appear in E by the corresponding terms t_i. Notice that if E does not contain any of the variables in the domain of θ, then applying θ to E leaves E unchanged. We say that $E \cdot \sigma$ is an *instance* of the expression E.

Definition 6 A substitution θ is *non-unifying* w.r.t a first order expressions E if for every pair of distinct first order terms $t, t' \in \mathit{Terms}(E)$ we have that $t \cdot \theta \neq t' \cdot \theta$.

We can prove the following:

Lemma 1 *Let θ (and subscripted variations of it) be substitutions, let S and s be two sets of atoms, b a single atom, and θ_N a non-unifying substitution w.r.t. $s \cup \{b\}$. Then,*

1. *If $b \in s$, then $b \cdot \theta \in s \cdot \theta$.*
2. *If $b \notin s$, then $b \cdot \theta_N \notin s \cdot \theta_N$.*

3. *If* $b \in S \setminus s$, *then* $b \cdot \theta \in S \cdot \theta \setminus s \cdot \theta$ *unless* $b \cdot \theta \in s \cdot \theta$.

4. *If* $b \in S \setminus s$, *then* $b \cdot \theta_N \in S \cdot \theta_N \setminus s \cdot \theta_N$.

5. *If* $\theta = (\theta_1 \cdot \theta_2)$ *and* $t \cdot \theta \neq t' \cdot \theta$, *then* $t \cdot \theta_1 \neq t' \cdot \theta_1$.

6. *If* $T \models s \to b$, *then* $T \models s \cdot \theta \to b \cdot \theta$.

Proof. We prove some of the properties, the rest are immediate. For Property 2, suppose that $b \notin s$. The substitution θ_N is non-unifying w.r.t. $s \cup \{b\}$, therefore, distinct terms in b remain distinct after applying θ_N. Therefore we can reverse θ_N, and we conclude that if $b \cdot \theta_N \in s \cdot \theta_N$ then $b \in s$. Hence, $b \cdot \theta_N \notin s \cdot \theta_N$. Notice that this is not necessarily true if the substitution involved is unifying. As an example, let $s = \{p(a)\}$, $b = p(x)$, and $\theta = \{x \mapsto a\}$. Then, $b \cdot \theta = p(a)$ and $s \cdot \theta = s$ since s does not contain x. Clearly, $b \notin s$ but $b \cdot \theta \in s \cdot \theta$. This is because θ has unified the terms x, a of $s \cup \{b\}$. Properties 2 and 3 imply Property 4. For Property 5, notice that if $t \cdot \theta_1 = t' \cdot \theta_1$, then θ cannot distinguish the terms t and t'. ■

The properties stated in the previous lemma are repeatedly used throughout the proof of the algorithm Learn-Closed-Horn in Chapter 5, although this is not always explicitly stated. Next, we describe a sound and complete deduction rule for first order Horn expressions.

Definition 7 A *derivation* of a clause $C = A \to a$ from a Horn expression T is a finite directed acyclic graph G with the following properties. Nodes in G are atoms possibly containing variables. The node a is the unique node of out-degree zero. For each node b in G, let $Pred(b)$ be the set of nodes b' in G with edges from b' to b. Then, for every node b in G, either $b \in A$ or $Pred(b) \to b$ is an instance of a clause in T. A derivation G of C from T is *minimal* if no proper subgraph of G is also a derivation of C from T.

Definition 8 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 $C \cdot \theta \subseteq D$.

Theorem 2 *Let T be any Horn expression and C be a non-trivial Horn clause. If $T \models C$, then there is a minimal derivation of C from T.*

Proof. First, we show that if we apply a substitution θ to all the nodes in a derivation graph G' (re-defining its edges accordingly) of some clause D from T then the resulting derivation graph which we denote $G' \cdot \theta$ is a derivation graph of $D \cdot \theta$ from T: consider any node b' in G'. If $Pred(b') \rightarrow b'$ is an instance of a clause in T, then $Pred(b') \cdot \theta \rightarrow b' \cdot \theta$ is too. If $b' \in Antecedent(D)$, then $b' \cdot \theta \in Antecedent(D \cdot \theta)$ as well. This covers all the nodes $b' \cdot \theta$ in $G \cdot \theta$ so we are done.

The Subsumption Theorem for SLD-derivation (Nienhuys-Cheng and De Wolf, 1997) guarantees that there is a SLD-derivation of C from T. Briefly, a SLD-derivation is a linear derivation $R_0 \rightarrow R_1 \rightarrow .. \rightarrow R_k = C'$ where C' is such that C' subsumes C, R_0 is a clause in T, and each R_i for $1 \leq i \leq k$ is obtained by resolving some clause C_i in T with the previous R_{i-1} using the consequent of C_i and a selected atom in the antecedent of R_{i-1} as the literals resolved upon.

Now we show how to transform any SLD-derivation of $R_k = C'$ from T into a derivation graph of C' from T by induction on the depth of the SLD-derivation k. If $k = 0$ then the SLD-derivation consists of just the clause C' in T. In this case, our derivation graph has as nodes all the literals in C' and edges $(l, Consequent(C'))$ for each literal $l \in Antecedent(C')$. Clearly this is a derivation graph for C' from T.

For the induction step ($k > 0$), suppose we have a derivation graph G_{k-1} of R_{k-1} from T. We show how to extend it to a derivation graph G_k of R_k from T. Let C_k be the clause in T that results in R_k by resolving it with R_{k-1}; let σ be the substitution that unifies the consequent of C_k and the selected atom $\hat{l}$ in the antecedent of R_{k-1}. Resolving $Consequent(C_k)$ with $\hat{l} \in Antecedent(R_{k-1})$ results in the clause $R_k = (Antecedent(R_{k-1}) \setminus \{\hat{l}\} \cup Antecedent(C_k)) \cdot \sigma \rightarrow Consequent(R_{k-1}) \cdot \sigma$. To obtain our desired derivation graph G_k, consider $G_{k-1} \cdot \sigma$ and add the literals in $Antecedent(C_k) \cdot \sigma$ as new nodes. Add the edges $(l \cdot \sigma, \hat{l} \cdot \sigma)$ for each literal $l \in Antecedent(C_k)$. Let the resulting graph be our G_k. Now, we claim that G_k is a derivation graph for R_k from T; it suffices to guarantee that the new/modified

nodes satisfy the conditions of a derivation graph. First, the node $\hat{l} \cdot \sigma$ satisfies that $Pred(\hat{l} \cdot \sigma) \rightarrow \hat{l} \cdot \sigma$ is an instance of a clause in T, in particular it is an instance of $C_k \in T$ since we added edges from all the literals in the antecedent of $C_k \cdot \sigma$ to $\hat{l} \cdot \sigma$ which is precisely the consequent of $C_k \cdot \sigma$. We have added new nodes $l \cdot \sigma$ for each $l \in Antecedent(C_k)$. Clearly, these nodes appear in the antecedent of R_k which contains $Antecedent(C_k) \cdot \sigma$.

Finally, to obtain the derivation graph of C from T, we just apply the substitution θ, where θ is the substitution s.t. $C' \cdot \theta \subseteq C$ to the derivation graph of $R_k = C'$ from T. By our remark above this is a valid derivation graph for $C' \cdot \theta$ which is also a valid derivation graph for C since $C' \cdot \theta \subseteq C$ and, more concretely, $Antecedent(C') \cdot \theta \subseteq Antecedent(C)$. ■

Definition 9 Let T be a first order Horn expression. Then T is *closed* if for any non-trivial clause C such that $T \models C$ it holds that all minimal derivations of C from T use first order terms appearing in C only.

Lemma 3 *Range restricted Horn expressions and constrained Horn expressions are closed.*

Proof. Range restricted Horn expressions: if b' appears in any derivation of $T \models s \rightarrow b$, where T is a range restricted Horn expression and s is a set of atoms, then obviously, $T \models s \rightarrow b'$. T is range restricted and therefore b' is made out of terms in s only. Thus, $b' \in Atoms_P(s) \subseteq Atoms_P(s \rightarrow b)$.

Constrained Horn expressions: consider any minimal derivation of $s \rightarrow b$ from a constrained Horn expression T. If b' appears in the derivation, then, since T is constrained, b' must be made out of terms in b only. Thus, $b' \in Atoms_P(b) \subseteq Atoms_P(s \rightarrow b)$. ■

2.3 The subsumption lattice

In this section we describe how the subsumption relation induces a lattice over the set of first order clauses. This establishes a clear notion of generality among clauses and it is very useful in visualizing the generalization operator that is used in the learning algorithm of Chapter 5.

2.3.1 Subsumption as a generality relation

We recall the definition of subsumption: 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 $D \preceq C$. C *strictly* or *properly subsumes* D, denoted $C \prec D$, if $C \preceq D$ but $D \not\preceq C$.

Definition 10 A relation $\preceq$ imposes a *quasi-order* on a set S if $\preceq$ is reflexive and transitive w.r.t. the elements in S.

Definition 11 A relation $\preceq$ imposes a *partial-order* on a set S if $\preceq$ is reflexive, anti-symmetric and transitive w.r.t. the elements in S.

Theorem 4 *If $C \preceq D$ then, $C \models D$.*

Proof. The Subsumption Theorem for SLD-derivation in (Nienhuys-Cheng and De Wolf, 1997) guarantees that there is a SLD-derivation of D from C. By (the proof of) Theorem 2, we know how to convert the SLD-derivation into a derivation graph of D from C. Soundness of derivation graphs (directly derived from the soundness of forward chaining) shows that $C \models D$. ■

Because of Theorem 4, we interpret $\preceq$ as a generality relation between clauses. The relation $\preceq$ is reflexive and transitive, and therefore it imposes a quasi-order on the set of first order clauses. However, this is not a partial order since $\preceq$ is not anti-symmetric: there exist clauses C_1, C_2 that are subsume-equivalent but are not identical, e.g., $C_1 = \{p(x,y), p(y,x), p(x,x)\}$ and $C_2 = \{p(x,x)\}$.

Clauses that can be obtained by renaming variables are considered identical. E.g., the clauses $C_3 = \{p(x, y), p(y, z)\}$ and $C_4 = \{p(x_1, x), p(x, z_3)\}$ are variable renamings, also called syntactic variants. Notice that in this case the variable renaming is given by $\{x \leftrightarrow x_1, y \leftrightarrow x, z \leftrightarrow z_3\}$.

The subsumption relation $\preceq$ and the set of first order clauses induce a lattice[2]. This is an important concept since generalizing or specializing a clause can be seen as moving up or down in the subsumption lattice.

2.3.2 Least general generalization as least upper bound

In the subsumption lattice, the least upper bound or *lub* of a pair of clauses C_1, C_2 is defined as a clause which is more general than both C_1 and C_2, and which is the least general such clause (w.r.t. subsumption). This is precisely computed by the *least general generalization* or *lgg* proposed by Plotkin (1970).

Definition 12 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:

[2]Strictly speaking, the relation $\preceq$ is a quasi-order and not a partial-order, so that $(Clauses, \preceq)$ does not induce a lattice in the standard set-theoretic sense. However, we relax the definition of lattice to work for quasi-orders which is enough for our purposes.

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 2 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

$$\{p(a, f(b)) - p(z, f(2)), \quad p(g(a, x), c) - p(z, f(2)), \quad q(a) - q(z)\}.$$

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 each clause if all the literals involved are compatible. In Chapter 5 we introduce the notion of a *pairing* which is a special subset of the *lgg* that avoids the explosion in size of the *lgg*. Pairings are a key aspect of our learning algorithm of Section 5.1. Notice that a pairing is more general than the *lgg* since it is a subset of the *lgg*; a pairing is therefore a generalization of the original pair of clauses, just not the minimal one.

Chapter 3

Learning From Queries

In this chapter we formalize our learning model. This involves formally defining the following: examples, concepts, types of queries available to the learning algorithms, and criterion of success of a learning algorithm.

Fix a signature $\mathcal{S} = (P, F)$; consider $\mathcal{FO}_{\mathcal{S}}$, the set of first order $\mathcal{S}$-expressions. We distinguish two different learning settings: *learning from interpretations* and *learning from entailment*.

Learning from interpretations. In this setting, examples are first order $\mathcal{S}$-interpretations. That is, interpretations must define a function mapping of the correct arity for every function symbol in F, and their extension must contain atoms built from predicates in P with the correct arity only. The universe of examples (all $\mathcal{S}$-structures) is noted by $\mathcal{I}_{\mathcal{S}}$.

A *concept* is a subset of $\mathcal{I}_{\mathcal{S}}$, i.e., a set of $\mathcal{S}$-interpretations. A concept $C \subseteq \mathcal{I}_{\mathcal{S}}$ is *represented* by a first order expression E if $I \models E \Leftrightarrow I \in C$, where $I \in \mathcal{I}_{\mathcal{S}}$. Notice that not all possible subsets of $\mathcal{I}_{\mathcal{S}}$ can be represented by first order expressions. In this thesis we consider concepts that can be represented by first order Horn expressions.

Learning from entailment. In this setting, examples are first order $\mathcal{S}$-clauses. The universe of examples is noted by $\mathcal{C}_\mathcal{S}$.

A *concept* is a subset of $\mathcal{C}_\mathcal{S}$, i.e. a set of first order clauses. A concept $C \subseteq \mathcal{C}_\mathcal{S}$ is *represented* by a first order expression E if $E \models c \Leftrightarrow c \in C$, where $c \in \mathcal{C}_\mathcal{S}$. In this thesis we consider concepts that can be represented by first order Horn expressions. Moreover, we restrict the universe of examples to Horn clauses only.

Parameterizing concept classes. In both cases (learning from interpretations and learning from entailment) we use $\mathcal{T}$ to refer to concept classes. In this thesis, concept classes are defined by restricting the types of first order expressions that are allowed. When the concept class is restricted to first order Horn expressions, we denote the concept class by $\mathcal{H}$ instead. We note that throughout this thesis we somehow blur the distinction between a class of concepts and the set of first order expressions representing the class.

Suppose that the function

$$Size : \mathcal{FO}_\mathcal{S} \longrightarrow \mathbb{N}^+$$

assigns to every first order expression a positive integer. Then, $Size(C)$, where C is a concept in some concept class $\mathcal{T}$, is defined as

$$Size(C) = min\left\{Size(R) \mid R \in \mathcal{FO}_\mathcal{S} \text{ and } R \text{ represents } C\right\}.$$

That is, the size of a concept is the size of the minimal first order expression representing it. Given a positive integer m, we define $\mathcal{T}^{Size \leq m}$ as the set of concepts represented by expressions of size at most m, i.e., $\mathcal{T}^{Size \leq m} = \{C \in \mathcal{T} \mid \ Size(C) \leq m\}$. When it is clear from the context what size we are referring to, we can write $\mathcal{T}^{\leq m}$. In Chapter 4 we study various notions of sizes for first order expressions in detail.

3.1 Queries

Assume that the target concept has been fixed, and that it is represented by some first order expression T. The query types we consider were introduced by Angluin (1988) and are:

Interpretation membership query. Given a first order interpretation $I \in \mathcal{I}_{\mathcal{S}}$, the query $MQ(I)$ returns `Yes` if $I \models T$ or `No` otherwise. The *input* to the query in this case is I.

Interpretation equivalence query. Given a first order expression H, the query $EQ(H)$ returns `Yes` if $H \equiv T$, otherwise it returns a *counterexample* $I \in \mathcal{I}_{\mathcal{S}}$ such that $I \models H$ and $I \not\models T$ or vice versa. That is, in case $H \not\equiv T$, the query returns an example proving this fact. The *input* to the query in this case is H.

Entailment membership query. Given a first order clause $c \in \mathcal{C}_{\mathcal{S}}$, the query $EntMQ(c)$ returns `Yes` if $T \models c$ or `No` otherwise. The *input* to the query in this case is c.

Entailment equivalence query. Given a first order expression H, the query $EntEQ(\cdot)$ returns `Yes` if $H \equiv T$, otherwise it returns a *counterexample* $C \in \mathcal{C}_{\mathcal{S}}$ such that $H \models C$ and $T \not\models C$ or vice versa. That is, in case $H \not\equiv T$, the query returns an example proving this fact. The *input* to the query in this case is H.

3.2 Computational complexity of queries

For completeness we include a partial survey of the computational power that is required from the oracles responding to the queries made by the algorithms. It is well known that if we do not restrict the expressions involved, oracles are required to solve undecidable problems! In our case, however, the use of closed Horn expressions makes all the queries decidable. Next, we list some of the problems (and their

computational complexity) associated with answering membership and equivalence queries that are of particular relevance to us. We assume that the inputs to our queries are both the target concept and the input to the query per se[1]. We assume that all the inputs to the queries as well as the target concept are finite.

On model checking. Checking whether $I \models C$ where I is a finite interpretation and C a clause is in general a decidable problem — one can exhaustively apply the rules of semantic satisfiability for first order expressions (see Section 2.2) and explore all combinations possible. However, this might be an expensive procedure. In fact, Vardi (1982) showed that the complexity of this problem is exponential in the size of C. Papadimitriou and Yannakakis (1997) refined this result and showed that the exponential dependence is in the number of variables in C rather than its total size. They show this by reducing *Clique* to the problem of deciding $I \models C$, where C is a range restricted function free Horn clause. Hence, answering interpretation membership queries, even for extremely simple target expressions, is at least NP-hard.

On single-clause implication. Schmidt-Schauss (1988) shows that checking whether $C \models D$ is undecidable if C and D are arbitrary clauses. More concretely, deciding $C \models D$ is semi-decidable in the sense that if the answer is `Yes` then we can always find a proof witnessing this fact. On the other hand, if the answer is `No` we might never know. Marcinkowski and Pacholski (1995) strengthen this result by proving that $C \models D$ remains semi-decidable, even if C and D are Horn. On the other hand, if C and D are datalog clauses[2], the problem becomes decidable; in particular, Gottlob and Papadimitriou (2003) show that the problem is EXPTIME-complete. In Section 5.1 we prove Theorem 14 stating that if closed Horn clauses

[1]This is what in the database theory is called "combined complexity" as opposed to "data complexity" and "expression complexity" where one assumes that the target concept is fixed, or that the input to the query is fixed, respectively. These complexities are quantitatively different as many results in database theory show.

[2]Datalog expressions are those containing terms that are either constants or variables.

are involved, the problem is decidable. Moreover, it is decidable with a polynomial number of subsumption tests (assuming constant arity).

On subsumption. Unfortunately, subsumption between Horn clauses is NP-complete (Kietz and Lübbe, 1994). However, Arimura (1997) shows that the subsumption problem if the Horn clauses are constrained is solvable in polynomial time; this is not hard to see since the only mapping from variables into terms one has to consider is the one dictated by the consequents of the clauses. It follows then from Theorem 14 in Section 5.1 that the implication problem for constrained Horn expressions is solvable in polynomial time as well. Finally, Khardon (1999b) shows that the implication problem for range restricted Horn clauses is decidable within exponential time. This result is implied by Theorem 14.

These results suggest that our models are too demanding: where can we find oracles to answer these rather difficult questions? Despite this, algorithms that learn from queries have been proved useful in practice in various ways. First, they give huge insights into the structure of the classes that the algorithms learn, thus allowing to exploit this structure in perhaps more practical scenarios. Second, some queries can be simulated easily: for example, equivalence queries can always be well approximated by using a polynomial-sized set of labeled examples (Angluin, 1988). In fact, equivalence queries can be seen as a useful abstraction of the learning scenario where labeled examples are available. Usually membership queries are a little harder to simulate, however, ad-hoc methods can be engineered in many cases to simulate these. Examples of successful query-based systems are (Shapiro, 1983; De Raedt and Bruynooghe, 1992; Reddy and Tadepalli, 1999; Khardon, 2000).

3.3 Models of learnability

In this thesis we use the model of exact learning from equivalence and membership queries. Other models using different types of queries, or models restricting the

possible queries to membership queries only or equivalence queries only are also possible. All these are reviewed in (Angluin, 1988; Angluin, 2001).

The following definitions assume that we have a notion of size for both first order expressions and examples. In case that examples are interpretations in $\mathcal{I}_{\mathcal{S}}$ (learning from interpretations), the size of an interpretation is defined as the number of elements in its domain. When examples are clauses (learning from entailment), then we use the same notion of size as for first order expressions (see Chapter 4).

Given some expression T in some concept class $\mathcal{T}$, a learning algorithm for $\mathcal{T}$ is required to output a hypothesis logically equivalent to T after asking a finite number of membership and equivalence queries. We use two complexity measures to parameterize the "goodness" of such a learning algorithm: the query complexity and the standard time complexity.

Definition 13 The *query complexity* of a learning algorithm $\mathcal{A}$ at any stage in a run is the sum of the sizes of the (i) inputs to equivalence queries, and (ii) inputs to membership queries made up to that stage. Notice that (i) refers to the size of first order expressions, and (ii) refers to the size of examples.

Definition 14 The *time complexity* of a learning algorithm is defined in the standard way, with queries taking just 1 time step, regardless of the sizes of the inputs to the queries. Nonetheless, if the algorithm makes an equivalence query with a very big hypothesis, its size is somehow accounted for in the time spent to construct it.

Finally, we define efficient learnability of a concept class:

Definition 15 A class $\mathcal{T}$ is *polynomial query-learnable* (*polynomial time-learnable*, resp.) if there exists a learning algorithm $\mathcal{A}$ and a two-variable polynomial $p(\cdot,\cdot)$ such that, for any positive integer m, and for any unknown target concept $T \in \mathcal{T}^{\leq m}$ all of the following hold:

(i) $\mathcal{A}$ uses membership queries and equivalence queries with hypotheses representing concepts in $\mathcal{T}$

(*ii*) $\mathcal{A}$ terminates and outputs an expression h representing the target T

(*iii*) at any stage, if n is the size of the longest counterexample received so far in response to an equivalence query, the query complexity (time complexity, resp.) of $\mathcal{A}$ at that stage does not exceed $p(n, m)$.

Chapter 4

Complexity of First Order Expressions

In this chapter we introduce different ways of quantifying the representation or description complexity of first order expressions (commonly referred to simply as *size*). Having a clear idea of what the different possible notions of size are and of how they interact seems crucial, since the main definitions of query and time complexity depend on the sizes that one uses, and hence affect the learning model directly. We relate these different description sizes using the notion of *polynomial relation*, which captures precisely those situations for which we can use interchangeably different sizes without changing the learning model.

4.1 Complexity measures

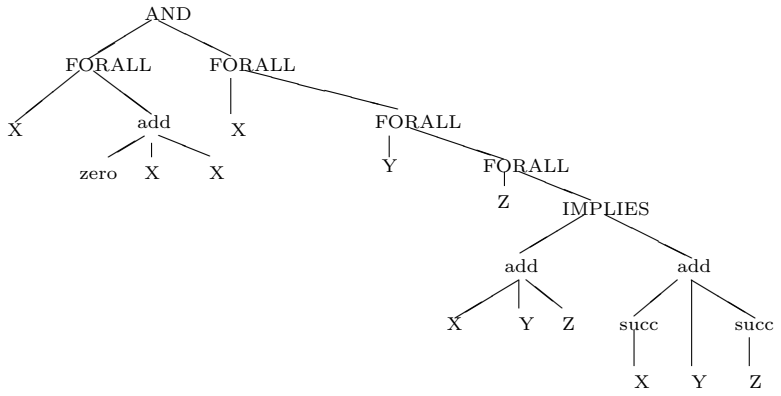
In this section we introduce all the complexity measures used throughout this document. We illustrate them using the following expression E:

$$(\forall X\ add(zero, X, X)) \wedge (\forall X\ \forall Y\ \forall Z\ add(X, Y, Z) \rightarrow add(succ(X), Y, succ(Z)))$$

***StringSize*(·):** as its name suggests, *StringSize* counts the number of syntactic symbols used to write down the input expression, ignoring spaces. Predicate and function symbols which use more than one letter contribute just 1. In our example, $StringSize(E) = 44$.

***WSize*(·):** similar to *StringSize*, *WSize* counts the number of syntactic symbols in the input expression, however, it does not count commas, parentheses or spaces. Function symbol occurrences contribute 2 (and hence its name — W comes from "weighted") and other symbol occurrences contribute just 1 to the total *WSize*. In our example, $WSize(E) = 27$.

***TreeSize*(·):** this size measure counts the number of nodes in a tree constructed recursively in the following manner. If the expression is a quantified expression, then put the quantifier in the root (labeled with the quantifier, `FORALL` or `EXISTS`), the quantified variable as its left child and the rest of the expression as the right child. If the expression is a conjunct, then add as children to the root (labeled with `AND`) all its conjuncts. Disjuncts are treated analogously, having `OR` as the root and the disjuncts as children. For implications the root is labeled with `IMPLIES` and the left child is the antecedent and the right child the consequent. With a negation the node is labeled with `NOT` and the only child is the rest of the expression. For atomic formulas, the root is labeled with the predicate symbol and the children are its arguments. If the expression is a variable, then the root is a leaf labeled with the variable name. For functional terms, the root is the outermost function symbol and the children are its arguments. In our example, $TreeSize(E) = 24$, and the associated tree is:



***DAGSize*(·):** counts the number of nodes in a DAG constructed by identifying identical subtrees in the tree constructed as explained above. We assume that expressions are *standardized apart*, that is, we avoid re-use of variable names that belong to scopes of different quantifiers. This converts our expression E into the equivalent E':

$$(\forall X'\ add(zero, X', X')) \wedge (\forall X\ \forall Y\ \forall Z\ add(X, Y, Z) \rightarrow add(succ(X), Y, succ(Z)))$$

In the example, the only repetition of terms are of variables X, Y, Z, X' which appear 3 times each. We save $4 \times (3-1) = 8$, hence $DAGSize(E') = TreeSize(E) - 8 = 16$.

***NTerms*(·):** if the input expression is a CNF expression, *NTerms* counts the maximum number of distinct terms (including sub-terms) appearing in any clause of the input expression. If the input expression E' is not a CNF, then *NTerms*(E') is exactly $|Terms(E')|$. In the example, $NTerms(E) = 5$, corresponding to term set in the second clause $\{X, Y, Z, succ(X), succ(Z)\}$. Throughout this document, we denote this parameter by t.

***WTerms*(·):** similar to the previously defined *NTerms*, with the only difference that functional terms are given twice as much weight as variables. In our example $WTerms(E) = 7$, corresponding to $\{X, Y, Z, succ(X), succ(Z)\}$.

***NVariables*(·):** if the input expression is a CNF expression, *NVariables* counts the maximum number of distinct variables appearing in any clause of the input expression. If the input expression E' is not a CNF, then $NVariables(E')$ is exactly $|Vars(E)|$. In the example, $NVariables(E) = 3$, corresponding to variable set in the second clause $\{X, Y, Z\}$. We denote this parameter by v.

***Depth*(·):** the maximum depth of any functional term appearing in the input expression. In the example, $Depth(E) = 2$ corresponding to the deepest term $succ(X)$ (or $succ(Z)$). We denote this parameter by d.

***NLiterals*(·):** if the input expression is a CNF expression, *NLiterals* equals the maximum number of literals in any clause of the input expression. Otherwise, it just counts the number of literals in the input expression. In the example, $NLiterals(E) = 2$ from the second clause. We denote this parameter by l.

***NPredicates*(·):** the number of distinct predicate symbols appearing in the input expression. In the example, $NPredicates(E) = 1$ corresponding to $\{add/3\}$. We denote this parameter by p.

***NFunctions*(·):** the number of distinct function symbols appearing in the input expression. In the example, $NFunctions(E) = 2$ corresponding to $\{zero/0, succ/1\}$. We denote this parameter by f.

***Arity*(·):** the largest arity of any predicate and function symbols appearing in the input expression. In the example, $Arity(E) = 3$ corresponding to the predicate $add/3$. We denote this parameter by a.

***NClauses*(·):** only defined for CNF expressions, *NClauses* equals the number of clauses in it. In our example, $NClauses(E) = 2$. We denote this parameter by c.

Size of meta-clauses

When quantifying the complexity of a meta-clause, we adopt a different approach:

Definition 16 Let *Size* be any complexity measure on first order expressions. Then $Size([s, c])$ is defined as the pair $(Size(s), Size(c))$.

Accordingly, we say that the meta-clause $[s_2, c_2]$ is more complex than the meta-clause $[s_1, c_1]$, denoted by $Size([s_1, c_1]) \leq Size([s_2, c_2])$, if $Size(s_1) < Size(s_2)$ or $(Size(s_1) = Size(s_2)$ and $Size(c_1) \leq Size(c_2))$. Also, $Size([s_1, c_1]) < Size([s_2, c_2])$, if $Size(s_1) < Size(s_2)$ or $(Size(s_1) = Size(s_2)$ and $Size(c_1) < Size(c_2))$.

4.2 Relating complexity measures

Definition 17 Let $\mathcal{C}$ be a class of first order expressions. Let k and j be positive integers. Let $\overline{C} = \{C_1, .., C_k\}$ be a list of complexity measures on expressions in $\mathcal{C}$, and let $\overline{D} = \{D_1, .., D_j\}$ be an alternative list of complexity measures on expressions in $\mathcal{C}$. We say that $\overline{C}$ and $\overline{D}$ are *polynomially related* w.r.t. $\mathcal{C}$ if there exist polynomials $p_1, .., p_k$ of arity j and polynomials $q_1, .., q_j$ of arity k such that for every $E \in \mathcal{C}$:

(i) for all $i = 1, .., k$: $C_i(E) \leq p_i(D_1(E), .., D_j(E))$, and

(ii) for all $i = 1, .., j$: $D_i(E) \leq q_i(C_1(E), .., C_k(E))$.

Lemma 5 *The polynomial relation between sets of complexity measures is reflexive, transitive, and symmetric.* ■

In the remainder of this section, we investigate which sets of complexity measures are polynomially related and which are not. Our main motivation in studying this problem comes from the discrepancy observed between the complexity measure used in the formal definitions of learnability (usually denoted by *Size*, without further explanation), and the complexity measures actually used by the algorithm developers in the literature (which use combinations of the following: *NTerms*, *NVariables*,

Depth, *NLiterals*, *NPredicates*, *NFunctions*, *Arity*, and *NClauses*). Here, we explore which of *TreeSize*, *DAGSize* *StringSize* and *WSize* are polynomially related to the set of alternative measures $\mathcal{M} = \{$*NTerms*, *NVariables*, *Depth*, *NLiterals*, *NPredicates*, *NFunctions*, *Arity*, *NClauses*$\}$.

4.2.1 Relating *StringSize* and *WSize*

Lemma 6 *StringSize is polynomially related to TreeSize.*

Proof. Let E be an arbitrary first order expression. Clearly, *TreeSize*$(E) \leq$ *StringSize*(E) since each node in the tree of E is counted by *StringSize*. To see that *StringSize*$(E) \leq p_1($*TreeSize*$(E))$ for some polynomial p_1, notice that the only syntactic objects that *StringSize* counts but *TreeSize* does not are parentheses and commas. First we account for the parentheses and commas due to function and predicate symbol applications. To each of the nodes in the tree of the arbitrary expression E we can charge a cost of 3 in the following way: if the node represents an atom or functional term, the root is charged an extra unit for the predicate symbol and the opening parenthesis, its children are charged with an extra unit for the commas, and the rightmost child for the ending parenthesis. The total of 3 comes from the fact that a child might be a functional term itself. Finally, we account for the parentheses due to expression grouping. To do this, we note that every time we use parentheses to group a subexpression, we are in fact "using up" some atom in the expression since otherwise it does not make sense to add parentheses. There are a maximum of *TreeSize*(E) atoms, so we can charge 2 extra units of cost to each atom in the tree (for opening and closing parentheses). Thus, *StringSize*$(E) \leq 5$ *TreeSize*(E). ■

Lemma 7 *WSize is polynomially related to TreeSize.*

Proof. Let E be an arbitrary first order expression. Clearly, *TreeSize*$(E) \leq$ *WSize*(E) since each node in the tree of E is certainly counted by *WSize*, in

some cases even twice. To see that $WSize(E) \leq p_1(TreeSize(E))$ for some polynomial p_1, notice that the only difference between *TreeSize* and *WSize* is that in *WSize* functional terms contribute 2 each instead of just 1 as in *TreeSize*. Thus, $WSize(E) \leq 2\ TreeSize(E)$. ■

Since the relation is symmetric and transitive, everything we say from now on about *TreeSize* is valid for *StringSize* as well as for *WSize*.

4.2.2 Relating *TreeSize*

The question now is whether we can find a combination of the alternative parameters in $\mathcal{M}$ that is polynomially related to *TreeSize*. Suppose that E is a first-order Horn expression s.t.

$$\begin{array}{lll} NTerms(E) = t & NVariables(E) = v & Depth(E) = d \\ NLiterals(E) = l & NPredicates(E) = p & NFunctions(E) = f \\ Arity(E) = a & NClauses(E) = c & \end{array}$$

Observe that any term appearing in E has size at most $O(a^d)$. Hence, any atomic formula has size at most $1 + O(a^{d+1}) = O(a^{d+1})$ (1 for the predicate symbol, a^{d+1} for the arguments). Hence, any Horn clause can have size no more than $1 + 2v + lO(a^{d+1}) = O(v + la^{d+1})$ (1 for the implication symbol in the clause, $2v$ for the quantifiers and quantified variables, and $O(a^{d+1})$ for each atom in the clause). Finally

$$TreeSize(E) = O(cv + cla^{d+1}).$$

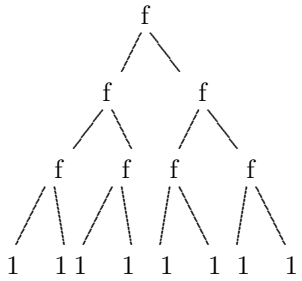
On the other hand, it is clear that all the parameters above are bounded by *TreeSize*(E). The next theorem shows that the converse does not hold:

Theorem 8 *TreeSize is not polynomially bounded by any combination of parameters that includes NTerms for classes over signatures with at least one constant and one function symbol of arity at least 2.*

Proof. We need to find some expression E such that its *TreeSize* is exponential in *NTerms*. Let $E = p(t_1)$, where t_1 is a complete tree of degree a with internal nodes labeled with function symbol f and leaves labeled with constant 1:

$$p(\overbrace{f(..f(f(f}^{d \text{ times}}(\overbrace{1,..,1}^{a \text{ times}}),..,f(1,..,1)),..,f(f(1,..,1),..,f(1,..,1)))..))$$

The following figure represents t_1 when $a = 2$, $d = 3$:



The complexity measures for E are:

$NTerms(E) = d$	$NVariables(E) = 0$	$Depth(E) = d$
$NLiterals(E) = 1$	$NPredicates(E) = 1$	$NFunctions(E) = 2$
$Arity(E) = a$	$NClauses(E) = 1$	$TreeSize(E) = \Theta(a^d)$

Hence no polynomial combination of the available complexity measures upper bounds $TreeSize(E)$. ■

This is a surprising fact that has not been noticed in previous work working with these parameters. No polynomial combination of the parameters above can replace *TreeSize*.

Lemma 9 *If we do not allow function symbols of arity greater than 1, then the set of parameters* $\{NClauses, NLiterals, Depth\}$ *is polynomially related to TreeSize.*

Proof. Follows from the fact that in this case $TreeSize = O(clad)$. ■

On the other hand, exponential lower bounds in terms of arity have been derived when ignoring *NLiterals*. These essentially reflect the following fact:

Lemma 10 *If the number of literals is ignored then TreeSize and DAGSize are not polynomially bounded by Arity*

Proof. Let p be a predicate of arity a. Let $\{1,\ldots,t\}$ be a set of t distinct terms built e.g. by one constant and one unary function. Let P be the set of all different $p()$ atoms built from these terms; $|P| = t^a$. Let $\hat{p}$ be a particular element in P. Let E be the expression $E = P \setminus \{\hat{p}\} \rightarrow \hat{p}$. The complexity of E is given by:

$NTerms(E) = t$	$NVariables(E) = 0$	$Depth(E) = t$
$NLiterals(E) = t^a$	$NPredicates(E) = 1$	$NFunctions(E) = 2$
$Arity(E) = a$	$NClauses(E) = 1$	
$TreeSize(E) = \Omega(t^a)$	$DAGSize(E) = \Omega(t^a)$	

Hence, the tree size is exponential in the arity when l is ignored. ■

4.2.3 Relating *DAGSize*

As in the case of *TreeSize*, *DAGSize* also gives an upper bound for all the alternative parameters in $\mathcal{M}$. This time the relation in the other direction is also polynomial. Notice that a DAG encodes terms in a smarter way, since multiple occurrences of a term are only counted once. Hence, t terms in a clause contribute $\Theta(t)$ to the *DAGSize* only. An atomic formula contributes only 1 since its arguments are encoded with the terms already. Hence, every clause has size at most $O(v+t+l) = O(t+l)$ and

$$c + l + t \leq DAGSize(E) = O(ct + cl).$$

Theorem 11 *The set of parameters $\{NTerms, NLiterals, NClauses\}$ is polynomially related to DAGSize w.r.t. the class of first order Horn expressions.* ■

Notice that the theorem is true for any values of the other parameters. The previous claim shows *DAGSize* can be exponential in arity but as the theorem shows in such a case one of c, l, t must be large as well. It is also interesting to note that several results on learning with queries, including ours in Chapter 5, give upper

bounds in terms of t^a and other parameters (Arimura, 1997; Reddy and Tadepalli, 1998; Rao and Sattar, 1998). While $l \leq p \cdot t^a$ these bounds do not directly relate to *DAGSize* or *TreeSize*.

4.3 Relating complexity measures and learning models

In this section we show that the notion of polynomial relation among complexity measures captures exactly the situations in which one can substitute the related complexity measures without changing the learning model (Lemma 12). For simplicity we assume that both examples and hypotheses are drawn from the same class, as it is, for example, in the case of learning from entailment. The result for the general case follows along similar lines.

Lemma 12 *Let $\mathcal{C}$ be a class of first-order expressions. Let $C_1, \ldots, C_k$ be a set of complexity measures that is polynomially related to Size w.r.t. the class $\mathcal{C}$, where Size is some notion of size for the expressions in $\mathcal{C}$. Let $p_1(\cdot), \ldots, p_k(\cdot)$ and $q(\cdot, \ldots, \cdot)$ be the polynomials witnessing their polynomial relation.*

Suppose that $\mathcal{A}$ is a learning algorithm for $\mathcal{C}$ with query complexity (w.r.t. alternative complexity measures $C_1, \ldots, C_k$) bounded by polynomials $s_i(c_1, \ldots, c_k, c'_1, \ldots, c'_k)$ for $i = 1, \ldots, k$, where $c_1, \ldots, c_k$ bound the complexity measures $C_1, \ldots, C_k$ for target concepts and $c'_1, \ldots, c'_k$ bounds the complexity measures for counterexamples received. Then, $\mathcal{A}$ is a learning algorithm for $\mathcal{C}$.

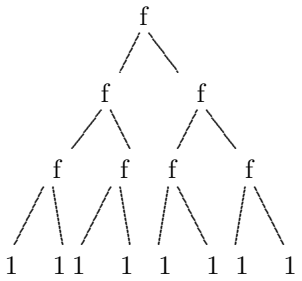
Proof. Notice that items (i) and (ii) from the definition of learnability hold trivially since we have assumed that $\mathcal{A}$ is a learning algorithm for $\mathcal{C}$ working in the same model. We show that item (iii) holds. Namely, there is a polynomial $r(\cdot, \cdot)$ s.t. at any stage, if n is the size of the longest counterexample received so far in response to an equivalence query, the query complexity of $\mathcal{A}$ at that stage does not exceed $r(n, m)$.

In the following, $\overline{f(args)}$ stands for $f_1(args), \ldots, f_k(args)$. We define $r(n, m)$ as $q(\overline{s(\overline{p}(m), \overline{p}(n))})$. Observe that all the functions $s_1, \ldots, s_k$, $p_1, \ldots, p_k$ and q are polynomials and hence r is a polynomial, too. It is left to show that r bounds the query complexity for $\mathcal{A}$.

Notice that $c \in \mathcal{C}_m$ implies that $c \in \mathcal{C}_{\overline{p}(m)}$ because $p_1(m), \ldots, p_k(m)$ bound the complexity measures in $C_1, \ldots, C_k$. By hypothesis, the query complexity (for complexity measures $C_1, \ldots, C_k$) of $\mathcal{A}$ is bounded by $\overline{s(\overline{p}(m), \overline{p}(n))}$. Hence, the query complexity of $\mathcal{A}$ is bounded by $q(\overline{s(\overline{p}(m), \overline{p}(n))})$. ■

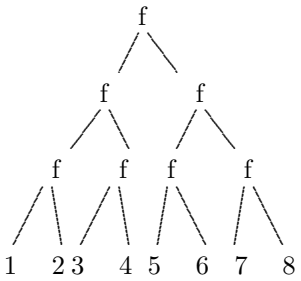
Remark 1 Note that we require polynomial bounds in both directions to guarantee learnability. This is needed for learning with queries and for proper PAC learnability (where hypothesis class is the same as concept class), whereas a one sided bound suffices for PAC predictability.

It is useful to highlight what can go wrong if this does not hold. In the figure below we can see three terms: t_1 has *TreeSize* exponential in the depth while its *DAGSize* is just linear; t_2 has both *TreeSize* and *DAGSize* exponential in the depth; finally t_3 has both *TreeSize* and *DAGSize* linear in the depth. Now, if one has an algorithm that learns w.r.t. *TreeSize* then when learning an expression including t_1 the algorithm is allowed to include t_2 in a query but this is not possible for learning w.r.t. *DAGSize* since t_1 is just polynomial in the depth whereas t_2 is exponential. On the other hand, if one has an algorithm that learns w.r.t. *DAGSize* then when learning an expression including t_3 the algorithm can use t_1 in its query. If we try to use this algorithm to learn w.r.t. *TreeSize* this query is too large.



t_1

$TreeSize(t_1) = \Theta(2^d)$
$DAGSize(t_1) = \Theta(d)$



t_2

$TreeSize(t_2) = \Theta(2^d)$
$DAGSize(t_2) = \Theta(2^d)$

t_3

$TreeSize(t_3) = \Theta(d)$
$DAGSize(t_3) = \Theta(d)$

Chapter 5

Learning Closed Horn Expressions

Here we present one of our main results: an algorithm that learns the class of Closed Horn Expressions. The learning algorithm described in this section generalizes the learning algorithm for the class of propositional Horn expressions (Frazier and Pitt, 1993) to first order Horn expressions and it inherits its high-level structure. Roughly speaking, our algorithm, like theirs, constructs hypotheses bottom-up — from specific to general — starting with the most specific hypothesis (the empty one) and further generalizing it by either adding new clauses or generalizing existing ones.

After every update of the hypothesis, the algorithm checks whether the current hypothesis is equivalent to the target expression by using the equivalence query oracle. If the answer is `Yes`, then the algorithm quits and returns the current hypothesis which is guaranteed to be correct. Otherwise, the algorithm uses the counterexample given by the oracle as part of the answer to further generalize the current hypothesis. This generalization occurs through two important mechanisms in the algorithm: *minimization* and *pairing*.

Minimization. Upon receipt of a counterexample from the equivalence query, this counterexample, which is a clause, is generalized by substituting complex terms by fresh variables and/or removing atoms from its antecedent while the clause is

still a counterexample — this can be checked with the membership query oracle. This operation is called minimization because the size of the clause that is being minimized is reduced.

Pairing. After the counterexample is minimized, the algorithm tries to combine it with existing clauses in the current hypothesis by constructing a pairing between them and checking whether the result is implied by the target — again, we use the membership query oracle to check this. A pairing is an operation based on the *least general generalization* or *lgg* (Plotkin, 1970; Plotkin, 1971); given two clauses it constructs a third clause which is strictly more general than the original ones. If the result of pairing a clause in the hypothesis with the counterexample is correct — i.e. implied by the target — then the clause participating in the pairing is replaced by the result of the pairing. If there is no successful pairing between the counterexample and some clause in the hypothesis, then the counterexample is appended to the current hypothesis.

Finally, the algorithm uses *meta-clauses* instead of standard clauses. Meta-clauses provide a compact way of representing sets of clauses that share the same antecedent. This allows a more efficient manipulation of the hypotheses, thus saving time and queries. This idea was used already by Angluin, Frazier, and Pitt (1992) to improve the complexity of their learning algorithm of propositional Horn Expressions.

5.1 The learning algorithm

Before describing the learning algorithm, we introduce some useful definitions. Suppose that the class $\mathcal{C}$ is closed. Suppose that $H, T \in \mathcal{C}$. Then we define:

- $\mathit{Cons\text{-}Closure}(T, [s, c]) = [s, \{b \in \mathit{Atoms}_P(s \cup c) \setminus s \mid T \models s \to b\}]$
- $\mathit{Ant\text{-}Closure}(H, [s, c]) = [\{b \in \mathit{Atoms}_P(s \cup c) \mid H \models s \to b\}, c]$

- $rhs(T, [s, c]) = \{b \in c \mid T \models s \rightarrow b\}$

Example 3 Let $T = \{p(x, y) \rightarrow q(x), q(x) \rightarrow r(x)\}$ and $H = \{p(x, x) \rightarrow r(x)\}$. Then,

- $\mathit{Cons\text{-}Closure}(T, [\{p(a, a)\}, \{q(b)\}]) = [\{p(a, a)\}, \{q(a), r(a)\}]$
- $\mathit{Ant\text{-}Closure}(H, [\{p(a, a)\}, \{q(b)\}]) = [\{p(a, a), r(a)\}, \{q(b)\}]$
- $rhs(T, [\{p(a, a)\}, \{q(b)\}]) = [\{p(a, a)\}, \{\}]$

The algorithm has to compute *Cons-Closure* and *rhs* for the case when T is the target expression only. Although it does not know what the target expression T is, it can use the *EntMQ* oracle to check, for appropriate atoms b, if $T \models s \rightarrow b$. Hence, the following algorithms compute these operations:

CONS-CLOSURE($[s, c]$)
1 $\mathit{CONS} \leftarrow \{b \in \mathit{Atoms}_P(s \cup c) \setminus s \mid \mathit{EntMQ}(s \rightarrow b) = \texttt{Yes}\}$
2 **return** $[s, \mathit{CONS}]$

RHS($[s, c]$)
1 $\mathit{CONS} \leftarrow \{b \in c \mid \mathit{EntMQ}(s \rightarrow b) = \texttt{Yes}\}$
2 **return** CONS

Notice that, in general, the set $\mathit{Ant\text{-}Closure}(H, [s, c])$ is not computable if H is not closed. However, in our case, we show that we can compute it with a polynomial number of subsumption tests by simple forward chaining. This is due to the fact that we only check for atoms in the polynomially bounded set $\mathit{Atoms}_P(s \cup c)$ as potential consequents. We incrementally construct the set of atoms in the antecedent (ANT in the following algorithm), starting with the initial antecedent s.

ANT-CLOSURE$(H, [s, c])$

1 $ANT \leftarrow s$

2 **repeat** **for** every atom b in $Atoms_P(s \cup c) \setminus ANT$

3 **if** clause $ANT \rightarrow b$ is subsumed by a clause $C \in H$

4 **then** $ANT \leftarrow ANT \cup \{b\}$

5 **until** no more atoms are added to ANT

6 **return** $[ANT, c]$

Lemma 13 *The algorithms* CONS-CLOSURE$([s, c])$, ANT-CLOSURE$(H, [s, c])$, *and* RHS$([s, c])$ *compute Cons-Closure*$(T, [s, c])$, *Ant-Closure*$(H, [s, c])$, *and rhs*$(T, [s, c])$ *respectively, where EntMQ is a membership entailment oracle for some target expression* T.

Proof. The correctness of the algorithms CONS-CLOSURE and RHS follows trivially from the assumption that the oracle is always correct. For the correctness of ANT-CLOSURE, take any atom $b \in$ *Ant-Closure*$(H, [s, c])$. By Theorem 2, there is a derivation of $s \rightarrow b$ from H. Moreover, H is closed and so is the derivation. Algorithm ANT-CLOSURE searches through all possible closed derivations systematically, therefore it eventually reaches the node b in the corresponding derivation, and b is included in the set ANT. Soundness of forward chaining guarantees that atoms not in *Ant-Closure*$(H, [s, c])$ are never added to the set ANT. ■

As a consequence, we obtain:

Theorem 14 *The problem of checking whether* $T \models C$, *where* T *is a closed Horn expression and* C *is a closed Horn clause, is decidable.* ■

Since subsumption can be solved in polynomial time for constrained Horn expressions (Arimura, 1997), we obtain the following:

Theorem 15 *The problem of checking whether* $T \models C$, *where* T *is a constrained Horn expression and* C *is a constrained Horn clause, is decidable in polynomial time.* ■

The situation with range restricted Horn expressions is different. For this type of clauses Vardi (1982) and Papadimitriou and Yannakakis (1997) show that the implication problem is NP-hard:

Theorem 16 *The problem of checking whether $T \models C$, where T is a range restricted Horn expression and C is a range restricted Horn clause, is NP-hard.* ■

We finally present our learning algorithm.

LEARN-CLOSED-HORN

1 $S \leftarrow [\,]; H \leftarrow \emptyset$
2 **while** *EntEQ*(H) returns $(\texttt{No}, A \rightarrow a)$
3 **do** $[s_x, c_x] \leftarrow$ *Minimize*$(H, A \rightarrow a)$
4 Find the first $[s_i, c_i] \in S$ s.t. $[s, c] \in$ *Basic-Pairings*$([s_x, c_x], [s_i, c_i])$
satisfies (i) $\text{RHS}([s, c]) \neq \emptyset$ and (ii) *WSize*$([s, c]) <$ *WSize*$([s_i, c_i])$
5 **if** such an $[s_i, c_i]$ is found
6 **then** replace it by the meta-clause $[s, \text{RHS}([s, c])]$
7 **else** append $[s_x, c_x]$ to S
8 $H \leftarrow \bigwedge_{[s,c] \in S} \{s \rightarrow b \mid b \in c\}$
9 **return** H

It remains to describe how to compute the operations *Minimize*$(H, A \rightarrow a)$ and *Basic-Pairings*$([s_x, c_x], [s_i, c_i])$.

5.1.1 Minimizing the counterexample

The minimization procedure transforms a counterexample clause $A \rightarrow a$ as generated by the equivalence query oracle into a more general meta-clause counterexample $[s_x, c_x]$. The following procedure implements *Minimize*$(H, A \rightarrow a)$:

MINIMIZE$(H, A \rightarrow a)$

1 $[s_x, c_x] \leftarrow$ CONS-CLOSURE(ANT-CLOSURE$(H, [A, \{a\}])$)

2 **for** every functional term t in $s_x \cup c_x$, in decreasing order of size

do Let $[s'_x, c'_x]$ be the meta-clause obtained from $[s_x, c_x]$ after substituting all occurrences of the term t by a new variable x_t

if RHS$(s'_x, c'_x) \neq \emptyset$

then $[s_x, c_x] \leftarrow [s'_x,$ RHS$(s'_x, c'_x)]$

3 **for** every term t in $s_x \cup c_x$, in increasing order of size

do Let $[s'_x, c'_x]$ be the meta-clause obtained after removing from $[s_x, c_x]$ all those atoms containing t

if RHS$(s'_x, c'_x) \neq \emptyset$

then $[s_x, c_x] \leftarrow [s'_x,$ RHS$(s'_x, c'_x)]$

4 **return** $[s_x, c_x]$

Example 4 This example illustrates the behavior of the minimization procedure. Parentheses are omitted; function f is unary. Suppose T consists of the single clause $p(a, fx) \rightarrow q(x)$, and the algorithm has received as counterexample the clause $p(a, f1), q(2), r(1) \rightarrow q(1)$. After step 1 of the minimization procedure, the counterexample is transformed into the (equivalent) meta-clause $[p(a, f1), q(2), r(1) \rightarrow q(1)]$. The next table shows the execution of the following loops in lines 2 and 3. The leftmost column shows the actual counterexample $[s_x, c_x]$ as it is being generalized. The middle column shows the term that is being generalized to a variable or that is being dropped. The rightmost column shows the resulting clause after the generalization $[s'_x, c'_x]$, with the implied atoms framed in a box.

$[s_x, c_x]$		*After generalizing term*
$[p(a, f1), q(2), r(1) \rightarrow q(1)]$	$f1 \mapsto X$	$[p(a, X), q(2), r(1) \rightarrow q(1)]$
$[p(a, f1), q(2), r(1) \rightarrow q(1)]$	$1 \mapsto X$	$[p(a, fX), q(2), r(X) \rightarrow \boxed{\text{q(X)}}]$
$[p(a, fX), q(2), r(X) \rightarrow q(X)]$	$2 \mapsto Y$	$[p(a, fX), q(Y), r(X) \rightarrow \boxed{\text{q(X)}}]$
$[p(a, fX), q(Y), r(X) \rightarrow q(X)]$	$a \mapsto Z$	$[p(Z, fX), q(Y), r(X) \rightarrow q(X)]$
$[s_x, c_x]$		*After dropping term*
$[p(a, fX), q(Y), r(X) \rightarrow q(X)]$	X	$[q(Y) \rightarrow]$
$[p(a, fX), q(Y), r(X) \rightarrow q(X)]$	Y	$[p(a, fX), r(X) \rightarrow \boxed{\text{q(X)}}]$
$[p(a, fX), r(X) \rightarrow q(X)]$	a	$[r(X) \rightarrow q(X)]$
$[p(a, fX), r(X) \rightarrow q(X)]$	fX	$[r(X) \rightarrow q(X)]$
$[p(a, fX), r(X) \rightarrow q(X)]$		

Notice that the minimized counterexample is very similar to the target clause. In fact, it is the case that every minimized counterexample contains as a subset a syntactic variant of one of the target clauses (Lemma 28). However, it may still contain extra atoms that the minimization procedure is unable to get rid of — like $r(X)$ in Example 4 — these have to disappear in some other way: *pairing*.

5.1.2 Pairing two meta-clauses

A crucial process in the algorithm is how two counterexamples are combined into a new one, hopefully yielding a better approximation of some target clause. The operation proposed here uses pairings of clauses, based on the *lgg*.

We have two meta-clauses, $[s_x, c_x]$ and $[s_i, c_i]$ that need to be combined. To do so, we generate a series of matchings between the terms of $s_x \cup c_x$ and $s_i \cup c_i$; each of these matchings produces a candidate to refine the sequence S.

Definition 18 A *matching* between two sets of terms T_x and T_i is a set $\sigma \subseteq T_x \times T_i$ that includes all the terms in one of the participating sets, i.e.: $|\sigma| = min(|T_x|, |T_i|)$.

Definition 19 A matching σ is 1-1 if terms are not re-used. Formally:

- For all $t_x \in T_x$ it holds that $|\{t' \mid (t_x, t') \in \sigma\}| \leq 1$, and
- For all $t_i \in T_i$ it holds that $|\{t' \mid (t', t_i) \in \sigma\}| \leq 1$.

Example 5 Let $T_x = \{a, b\}$ and $T_i = \{1, 2, f(1)\}$. Notice that pairs are not denoted by the usual notation (a, b) but by $a - b$. The possible 1-1 matchings are:

$$\sigma_1 = \{a-1, b-2\} \qquad \sigma_3 = \{a-2, b-1\} \qquad \sigma_5 = \{a-f(1), b-1\}$$
$$\sigma_2 = \{a-1, b-f(1)\} \quad \sigma_4 = \{a-2, b-f(1)\} \quad \sigma_6 = \{a-f(1), b-2\}$$

Definition 20 An *extended matching* is an ordinary matching with an extra element added to every entry of the matching. This extra element contains the *lgg* of every pair in the matching. The *lgg*s are simultaneous, that is, they share the same table.

Definition 21 An extended matching σ is *legal* if every subterm of some term appearing as the *lgg* of some entry, also appears as the *lgg* of some other entry of σ. An ordinary matching is legal if its extension is.

Example 6 Parentheses are omitted as functions f and g are unary. Let σ_1 be $\{a - c, fa - b, ffa - fb, gffa - gffc\}$ and $\sigma_2 = \{a - c, fa - b, ffa - fb\}$. The matching σ_1 is not legal, since the term fX is not present in its extension column and it is a subterm of $gffX$, which is present. The matching σ_2 is legal.

Extended σ_1	*Extended* σ_2
`[a - c => X]`	`[a - c => X]`
`[fa - b => Y]`	`[fa - b => Y]`
`[ffa - fb => fY]`	`[ffa - fb => fY]`
`[gffa - gffc => gffX]`	

Our algorithm considers yet a more restricted type of matching.

Definition 22 A *basic matching* σ is a 1-1, legal matching between two sets T_x and T_i. This operation is asymmetric and the order in which the arguments is given

is relevant. It is only defined if $|T_x| \leq |T_i|$, where T_x is the first argument and T_i the second. It restricts how the functional structure of the terms is matched. Formally, if entry $f(t_1, .., t_n) - t \in \sigma$, then $t = f(r_1, .., r_n)$ and $t_i - r_i \in \sigma$ for all $i = 1, .., n$.

As we show below, a basic matching maps all variables in T_x to terms in T_i and then adds the remaining entries following the functional structure of the terms in T_x. Therefore an entry $x - f(y)$ might be included in a basic pairing but an entry $f(y) - x$ cannot. The following procedure shows how to construct basic matchings between sets of terms T_x and T_i.

BASIC-MATCHINGS(T_x, T_i)

1 Match every *variable* in T_x to a different *term* in T_i. Every possibility potentially yields a basic matching between T_x and T_i

2 Complete all potential basic matchings by adding the functional terms in T_x to the basic matchings as follows:

for every potential basic matching created in step 1

do Consider all functional terms in T_x in an upwards fashion, beginning with simpler terms:

for every term $f(t_1, .., t_n)$ in T_x such that all $t_i - r_i$ (with $i = 1, .., n$) appear in the basic matching already

do Add a new entry $f(t_1, .., t_n) - f(r_1, .., r_n)$

if $f(r_1, .., r_n)$ does not appear in T_i or the term $f(r_1, .., r_n)$ has been used already

then discard the matching

Example 7 Let $T_x = \{a, x, fx\}$ and $T_i = \{a, 1, 2, f1\}$. No parentheses for functions are written. The algorithm starts by matching variables in T_x to terms in T_i. Then, it matches functional terms in T_x using the constraints described in the procedure above. This computation is described in the table below.

Terms	*Matching 1*	*Matching 2*	*Matching 3*	*Matching 4*
x	$x-a$	$x-1$	$x-2$	$x-f1$
a	NO! $a-a$	$a-a$	$a-a$	$a-a$
fx	DISCARDED	$fx-f1$	NO! $fx-f2$	NO! $fx-ff1$
	DISCARDED	OK	DISCARDED	DISCARDED

The table is interpreted as follows. In the first column we have the terms in T_x in the order considered by our algorithm. In the columns thereafter, we have all potential matchings. The last row indicates which of the matchings has been discarded. The entries on top of the "OK" matchings contain the matching's pairs.

Notice that we have only 1 basic matching between the set of terms $\{a, x, fx\}$ and $\{a, 1, 2, f1\}$. Compare this with the 24 different 1-1 matchings that would be considered by previous algorithms. This difference grows with the complexity of the functional structure in the examples.

Lemma 17 BasicMatchings(T_x, T_i) *finds all basic matchings.*

Proof. First, we show that every matching constructed by the procedure is basic. It is 1-1 because after step 1 the matchings are 1-1, and the new pairs added in step 2 are checked not to be included in the matchings already. It is legal because only terms which have all of its subterms included in the matching are added. It is basic because functional structure is respected when adding a new pair.

Second, we show that every basic matching is found by the procedure. First notice that matchings including the combination of a pair (functional term in T_x, variable in T_i) is not permitted, since subterms of the functional term in T_x have to be included in the matching and they would not have any possible legal term to be matched to because a variable has no subterms. Therefore, the only possibility involving variables is (variable in T_x, term in T_i). All these are found in step 1 of the procedure and appropriately completed in step 2. ■

One of the key points of our algorithm lies in reducing the number of matchings that need to be checked by ruling out some of the candidate matchings that do not

satisfy the restrictions imposed. By doing so we avoid testing too many pairings and hence avoid making unnecessary calls to the oracles. One of the restrictions has already been mentioned, it consists in considering basic pairings only, as opposed to considering every possible matching. Let t be an upper bound on the number of terms in T_x and T_i, and let v be an upper bound on the number of variables in T_x and T_i. There are t^t possible distinct matchings but only t^v distinct *basic* pairings: we only combine *variables* of T_x with *terms* in T_i. The other restriction on the candidate matching consists in the fact that every one of its entries must appear in the original *lgg* table, as we are going to see shortly.

Given two meta-clauses $[s_x, c_x]$ and $[s_i, c_i]$, the idea is to first compute the set of basic matchings as given by BASIC-MATCHINGS($Terms(s_x \cup c_x)$, $Terms(s_i \cup c_i)$). Each of these basic matchings computed determines then a distinct pairing between the meta-clauses $[s_x, c_x]$ and $[s_i, c_i]$.

Pairing is an operation that takes two meta-clauses and a matching between its terms and produces another meta-clause. We say that the pairing is *induced* by the matching it is fed as input. A *legal pairing* is a pairing for which the inducing matching is legal; a *basic pairing* is one for which the inducing matching is basic.

The antecedent s of the pairing is computed as the *lgg* of s_x and s_i restricted to the matching σ inducing it; we denote this by $lgg_{|\sigma}(s_x, s_i)$. An atom is included in the pairing only if all of its top-level terms appear as entries in the extended matching. This restriction is quite strong in the sense that, for example, if an atom $p(f(x))$ appears in both s_x and s_i then their *lgg* $p(f(x))$ is not included unless the entry `[f(x) - f(x) => f(x)]` appears in the matching. In case `[x - x => x]` appears but `[f(x) - f(x) => f(x)]` does not, the atom $p(f(x))$ is ignored. We only consider matchings that are subsets of the *lgg* table.

The consequent c of the pairing is computed as the union of the sets $lgg_{|\sigma}(s_x, c_i)$, $lgg_{|\sigma}(c_x, s_i)$ and $lgg_{|\sigma}(c_x, c_i)$. Note that in the consequent all the possible *lgg*s of pairs among $\{s_x, c_x\}$ and $\{s_i, c_i\}$ are included except $lgg_{|\sigma}(s_x, s_i)$, which constitutes the antecedent.

When computing any of the *lgg*s, the same table is used. That is, the same pair of terms is bound to the same expression in any of the four possible *lgg*s that are computed in a pairing. The pairing between $[s_x, c_x]$ and $[s_i, c_i]$ induced by σ is computed as follows:

PAIRING$(\sigma, [s_x, c_x], [s_i, c_i])$

1 $s \leftarrow lgg_{|\sigma}(s_x, s_i)$

2 $c \leftarrow lgg_{|\sigma}(s_x, c_i) \cup lgg_{|\sigma}(c_x, s_i) \cup lgg_{|\sigma}(c_x, c_i)$

3 **return** $[s, c]$

Finally, we describe the algorithm that computes *Basic-Pairings*$([s_x, c_x], [s_i, c_i])$, the set of basic pairings between two meta-clauses $[s_x, c_x]$ and $[s_i, c_i]$:

BASIC-PAIRINGS$([s_x, c_x], [s_i, c_i])$

1 $PAIRINGS = \emptyset$

2 **for** each $\sigma \in$ BASIC-MATCHINGS$(Terms(s_x \cup c_x), Terms(s_i \cup c_i))$

3 **do if** $\sigma \subseteq lgg\text{-}table(s_x \cup c_x, s_i \cup c_i)$

4 **then** $PAIRINGS \leftarrow PAIRINGS \cup \{$PAIRING$(\sigma, [s_x, c_x], [s_i, c_i])\}$

5 **return** *PAIRINGS*

Example 8 The table below describes two examples. Both examples have the same terms as in Example 7, so there is only one basic matching. Ex. 8.1 shows how to compute a pairing. Ex. 8.2 shows that a basic matching may be rejected if it does not agree with the *lgg* table (entries `[x - 1 => X]` and `[fx - f1 => fX]` do not appear in the *lgg* table).

	Example 8.1	*Example 8.2*
s_x	$\{p(a, fx)\}$	$\{p(a, fx)\}$
s_i	$\{p(a, f1), p(a, 2)\}$	$\{q(a, f1), p(a, 2)\}$
$lgg(s_x, s_i)$	$\{p(a, fX), p(a, Y)\}$	$\{p(a, Y)\}$
lgg table	`[a - a => a]`	`[a - a => a]`
	`[x - 1 => X]`	`[fx - 2 => Y]`
	`[fx - f1 => fX]`	
	`[fx - 2 => Y]`	
basic σ	`[a-a=>a]`	`[a-a=>a]`
	`[x-1=>X]`	`[x-1=>X]`
	`[fx-f1=>fX]`	`[fx-f1=>fX]`
$lgg_{\|\sigma}(s_x, s_i)$	$\{p(a, fX)\}$	PAIRING REJECTED

As the examples demonstrate, the requirement that the matchings are both basic and comply with the *lgg* table is quite strong. The more structure examples have, the greater the reduction in possible pairings (and hence queries), since that structure needs to be matched. While it is not possible to quantify this effect without introducing further parameters, we expect this to be a considerable improvement in practice.

A note for potential implementations In practice, when trying to construct basic pairings between s_x and s_i it is better to consider as entries for the matching those entries appearing in the *lgg* table only. That is, when combining meta-clauses $[s_x, c_x]$ and $[s_i, c_i]$, one would first compute the $lgg(s_x, s_i)$ and record the *lgg* table. The next step would be to construct basic pairings using the entries in the *lgg* table. Instead of considering any pair between terms of s_x and s_i, the choice would be restricted to those pairs of terms present in the *lgg* table. The advantage of this method is that subsets of the *lgg* table that constitute a basic matching are systematically constructed. This implies that there is no need to check whether a given basic matching agrees with the *lgg* table and only subsets of the *lgg* table

are generated. This consideration is not reflected in the bounds for the worst case analysis. However, it should constitute an important speedup in practice.

5.2 Proof of correctness

Before going into the details of the proof of correctness, we describe the transformation $U(T)$ performed on a target expression T. It extends the transformation described by Khardon (1999a) (where expressions were function-free) and it serves analogous purposes.

5.2.1 Transforming the target expression

This transformation is never computed by the learning algorithm; it is only used in the analysis. The transformation introduces new clauses and adds some inequalities to every clause's antecedent. This avoids unification of terms in the transformed clauses. Related work by Semeraro et al. (1998) also uses inequalities in clauses, although the learning algorithm and approach are completely different.

The idea is to create a new set of clauses $U(C)$ from every clause C in T. Every clause in $U(C)$ corresponds to the original clause C with its terms unified in a unique way, different from every other clause in $U(C)$. Every possible unification of terms of C are covered by one of the clauses in $U(C)$. The clauses in $U(C)$ are only satisfied if the terms are unified in exactly that way.

$U(T)$

1 $U \leftarrow \emptyset$

2 **for** every clause $C = s_c \rightarrow b_c$ in T

3 **do for** every partition $\pi = \{\pi_1, \pi_2, .., \pi_l\}$ of $Terms(C)$

4 **do** $A_\pi \leftarrow \{A(t_1, .., t_l) \mid t_i \in \pi_i, \text{for all } i = 1, .., l\}$

5 Let σ_π be an *mgu* of A_π

6 **if** no *mgu* exists or there exist $i \neq j$ s.t. $\pi_i \cdot \sigma_\pi = \pi_j \cdot \sigma_\pi$

7 **then** discard the partition

8 **else** $U_\pi(C) \leftarrow ineq(C \cdot \sigma) \wedge s_c \cdot \sigma \rightarrow b_c \cdot \sigma$

9 $U \leftarrow U \wedge U_\pi(C)$

10 **return** U

We construct $U(T)$ from T by considering every clause separately. For a clause C in T we generate a set of clauses $U(C)$. To do that, we consider all partitions of the set of terms in $Terms(C)$; each such partition, say π, can generate a clause of $U(C)$, denoted $U_\pi(C)$. Therefore, $U(T) = \bigwedge_{C \in T} U(C)$ and $U(C) = \bigwedge_{\pi \in ValidPartitions(Terms(C))} U_\pi(C)$. The set $ValidPartitions(Terms(C))$ captures those partitions for which a simultaneous unifier of all of its classes exists and whose representatives are all different. The use of A_π provides the simultaneous *mgu*; uniqueness of representatives is tested on line 6 in the transformation algorithm. We call a *representative* of a class π_i the only element in $\pi_i \cdot \sigma_\pi$, where σ_π is an *mgu* for the set A_π as described in the algorithm above.

Example 9 Let C be $p(f(x), f(y), g(z)) \rightarrow q(x, y, z)$. The terms appearing in C are $\{x, y, z, f(x), f(y), g(z)\}$. We consider some possible partitions:

- When $\pi = \{x, y\}, \{z\}, \{f(x), f(y)\}, \{g(z)\}$, then

$$A_\pi = \begin{cases} A(x, z, f(x), g(z)) \\ A(x, z, f(y), g(z)) \\ A(y, z, f(x), g(z)) \\ A(y, z, f(y), g(z)) \end{cases}$$

An *mgu* for A_π is $\sigma_\pi = \{y \mapsto x\}$. Therefore,

$$U_\pi(C) = (x \neq z \neq f(x) \neq g(z)), p(f(x), f(x), g(z)) \rightarrow q(x, x, z).$$

- When $\pi' = \{x, y, z\}, \{f(x), g(z)\}, \{f(y)\}$, then

$$A_{\pi'} = \begin{cases} A(x, f(x), f(y)) \\ A(x, g(z), f(y)) \\ A(y, f(x), f(y)) \\ A(y, g(z), f(y)) \\ A(z, f(x), f(y)) \\ A(z, g(z), f(y)) \end{cases}$$

  There is no *mgu* for the set $A_{\pi'}$, therefore this partition does not contribute to the transformation $U(C)$.

- When $\pi'' = \{x, y\}, \{z\}, \{f(x)\}, \{f(y)\}, \{g(z)\}$, then

$$A_{\pi''} = \begin{cases} A(x, z, f(x), f(y), g(z)) \\ A(y, z, f(x), f(y), g(z)) \end{cases}$$

  An *mgu* for $A_{\pi''}$ is $\sigma_{\pi''} = \{y \mapsto x\}$. However, this partition is discarded because the representatives for classes π_3 and π_4 coincide: $\pi_3 \cdot \sigma_\pi = \{f(x)\} = \pi_4 \cdot \sigma_\pi$. Notice that the partition π covers the case when the terms $f(x)$ and $f(y)$ are unified into the same term, so adding this clause would introduce repeated clauses in the transformation.

We write the fully inequated clause "$ineq(s_t \rightarrow b_t) \wedge s_t \rightarrow b_t$" as "$s_t \xrightarrow{\neq} b_t$". The following facts hold for T and its transformation $U(T)$.

Lemma 18 *If an expression T has c clauses, then the number of clauses in its transformation $U(T)$ is at most ct^v, where t (v, resp.) is the maximum number of different terms (variables, resp.) in any clause in T.*

Proof. It suffices to see that any clause C produces at most t^v clauses in $U(C)$. We show that if π and π' are two partitions that are not discarded by the transformation algorithm and $\sigma_\pi = \sigma_{\pi'}$, then $\pi = \pi'$. Suppose, then, that π and π' are two successful partitions such that $\sigma_\pi = \sigma_{\pi'}$. Let t and t' be two distinct terms of C in the same class in π. Notice that since σ_π is a unifier for A_π, t and t' have the same representative. Therefore, these two terms have to fall into the same class in π' (otherwise π' would be rejected). Since the same argument also holds in the opposite direction (i.e. from π' to π) we conclude that for all terms t, t' of C, t and t' are placed in the same class in π if and only they are placed in the same class in π'. Hence, $\pi = \pi'$. Finally, the bound follows since there are at most t^v substitutions mapping the at most v variables into the at most t terms. ■

Lemma 19 *$T \models U(T)$.*

Proof. Follows from the fact that every clause in $U(T)$ is subsumed by the clause in T that originated it. ■

Corollary 20 *If $U(T) \models C$, then $T \models C$. Also, if $U(T) \models [s, c]$, then $T \models [s, c]$.*

However, the inverse implication $U(T) \models T$ of Lemma 19 does not hold. To see this, consider the following example.

Example 10 We present an expression T, its transformation $U(T)$ and an interpretation I such that $I \models U(T)$ but $I \not\models T$. The expression T is $\{p(a, f(a)) \rightarrow q(a)\}$ and its transformation $U(T) = \{(a \neq f(a)), p(a, f(a)) \rightarrow q(a)\}$. The interpretation I has domain $D_I = \{1\}$; the only constant $a = 1$; the only function $f(1) = 1$ and the extension $ext(I) = \{p(1, 1)\}$.

$I \not\models T$ because $p(a, f(a))$ evaluates to 1 under I but $q(a)$ evaluates to 0.

$I \models U(T)$ because inequality $(a \neq f(a))$ evaluates to 0 and therefore the antecedent of the clause is falsified. Hence, the clause is satisfied.

5.2.2 Some definitions and notation

During the analysis, p stands for the cardinality of P, the set of predicate symbols in the language; a for the maximal arity of the predicates in P; v for the maximum number of distinct variables in a clause of T; t for the maximum number of distinct terms in a clause of T; t' for the maximum number of distinct terms in a counterexample; c for the number of clauses of the target expression T; c' for the number of clauses of the transformation of the target expression $U(T)$.

Definition 23 A meta-clause $[s, c]$ *covers* a fully-inequated clause $s_t \xrightarrow{\neq} b_t$ if there exists a mapping θ from variables in $s_t \cup \{b_t\}$ into terms in $Terms(s \cup c)$ such that the following three conditions are satisfied:

- $s_t \cdot \theta \subseteq s$
- $ineq(s_t \cup \{b_t\}) \cdot \theta \subseteq ineq(s \cup c)$
- $b_t \cdot \theta \in Atoms_P(s \cup c)$.

The condition $ineq(s_t \cup \{b_t\}) \cdot \theta \subseteq ineq(s \cup c)$ establishes that the substitution θ is non-unifying, i.e., it does not unify terms in $s_t \rightarrow b_t$ in the sense that two distinct terms in $s_t \rightarrow b_t$ remain distinct after applying the substitution θ.

Definition 24 A meta-clause $[s, c]$ *captures* a clause $s_t \xrightarrow{\neq} b_t$ if $[s, c]$ covers $s_t \xrightarrow{\neq} b_t$ and, in addition, $b_t \cdot \theta \in c$, for some θ witnessing the fact that $[s, c]$ covers $s_t \xrightarrow{\neq} b_t$.

Definition 25 A meta-clause $[s, c]$ is *correct* w.r.t. an expression T if $T \models [s, c]$, i.e. $T \models s \rightarrow b$ for all $b \in c$.

Definition 26 A meta-clause $[s, c]$ is *complete* w.r.t. an expression T if $b \in c$ for all atoms $b \in Atoms_P(s \cup c) \setminus s$ s.t. $T \models s \rightarrow b$.

Definition 27 A meta-clause $[s, c]$ is *full* w.r.t. an expression T if it is correct and complete w.r.t. T.

Example 11 Let $T = \{(x \neq y), p(x, y) \rightarrow q(x), q(x) \rightarrow r(x)\}$. Then,

- $[p(a, b) \rightarrow q(a)]$ covers $(x \neq y), p(x, y) \rightarrow q(x)$ with $\theta = \{x \mapsto a, y \mapsto b\}$.
- $[p(a, b) \rightarrow q(a)]$ captures $(x \neq y), p(x, y) \rightarrow q(x)$ with $\theta = \{x \mapsto a, y \mapsto b\}$.
- $[p(a, a) \rightarrow q(a)]$ does not cover $(x \neq y), p(x, y) \rightarrow q(x)$ because x and y are unified and hence $x \neq y$ does not hold.
- $[p(a, b) \rightarrow r(a)]$ does not cover $(x \neq y), p(x, y) \rightarrow q(x)$ because there is no θ such that $q(x) \cdot \theta$ appears in the meta-clause.
- $[p(a, b), q(a) \rightarrow r(a)]$ covers $(x \neq y), p(x, y) \rightarrow q(x)$.
- $[p(a, b), q(a) \rightarrow r(a)]$ does not capture $(x \neq y), p(x, y) \rightarrow q(x)$ because $q(a)$ is not in the consequent.
- $[p(a, b) \rightarrow r(a)]$ is correct w.r.t. T.
- $[p(a, b) \rightarrow r(b)]$ is not correct w.r.t. T.
- $[p(a, b) \rightarrow r(a)]$ is not complete w.r.t. T because $q(a)$ is missing.
- $[p(a, b), q(a) \rightarrow r(a)]$ is complete w.r.t. T.
- $[q(a) \rightarrow r(a)]$ is complete w.r.t. T.
- $[p(a, b) \rightarrow q(a), r(a)]$ is complete w.r.t. T.
- $[p(a, b) \rightarrow q(a), r(a)]$ is full w.r.t. T.
- $[p(a, b) \rightarrow r(b)]$ is not full w.r.t. T because it is not correct w.r.t. T.
- $[p(a, b) \rightarrow r(a)]$ is not full w.r.t. T because it is not complete w.r.t. T.

5.2.3 Brief description of the proof of correctness

If the algorithm stops, then the returned hypothesis is correct. Therefore we focus our attention in proving that the algorithm finishes. To do so, a bound is established on the length of the sequence S, that is, only a finite number of counterexamples can be added to S. Since every refinement of an existing meta-clause reduces its size, termination is guaranteed.

To bound the length of the sequence S the following condition is proved. Every element in S captures some clause of $U(T)$ but no two distinct elements of S capture the same clause of $U(T)$ (Lemma 46). The bound on the length of S is therefore c', the number of clauses of the transformation $U(T)$.

To see that every element in S captures some clause in $U(T)$, it is shown that all counterexamples in S are full meta-clauses w.r.t. the target expression T (Lemma 37) and that any full meta-clause must capture some clause in $U(T)$ (Lemma 22).

To see that no two distinct elements of S capture the same clause of $U(T)$, two important properties are established in the proof. First, Lemma 38 shows that if a counterexample $[s_x, c_x]$ captures some clause of $U(T)$ which is covered by some $[s_i, c_i]$ then the algorithm replaces $[s_i, c_i]$ with one of their basic pairings. Second, Lemma 35 shows that a basic pairing cannot capture a clause not captured by either of the original clauses. These properties are used in Lemma 46 to prove uniqueness of captured clauses.

Once the bound on S is established, we derive our final theorem by carefully counting the number of queries made to the oracles in every procedure.

Notation. Throughout the proof, we adopt the following conventions. We use the letter s and subscripted variants of it (like s_t, s_x, etc.) to denote sets of atoms constituting antecedent of clauses or meta-clauses. Similarly c and subscripted variants denote sets of atoms forming the consequents of meta-clauses. The letter b and its subscripted variants denote a single atom, and are mostly used as in

the consequent of a clause, e.g. as in $s_t \rightarrow b_t$. T refers to some arbitrary Closed Horn Expression assumed to be the hidden target concept. Arbitrary clauses in the target T are noted as $s_t \rightarrow b_t$, and $s_t \stackrel{\neq}{\rightarrow} b_t$ denotes some fully-inequated variant of the clause $s_t \rightarrow b_t$ appearing in $U(s_t \rightarrow b_t)$. The letter t refers to two things (hopefully it is clear from the context which one is referred to each time). It is used to either denote the upper bound on the number of terms in each clause of T, or to denote arbitrary terms occurring in clauses and meta-clauses (together with variants of it). H stands for the hypotheses constructed during the execution of the learning algorithm Learn-Closed-Horn. The meta-clause $[s_x, c_x]$ refers to the result of minimizing a counterexample, although when arguing about the behavior of Minimize, it also refers to the counterexample in intermediate stages of the process. The meta-clause $[s_i, c_i]$ denotes any of the meta-clauses that are added to the sequence S of the algorithm Learn-Closed-Horn. Finally, $[s, c]$ refers to some basic pairing between the minimized counterexample $[s_x, c_x]$ and some meta-clause $[s_i, c_i] \in S$, and it is also used to denote arbitrary meta-clauses. We proceed with the analysis in detail.

5.2.4 Properties of full meta-clauses

Lemma 21 *If C subsumes $[s, c]$, then $[s, c]$ captures some clause in $U(C)$.*

Proof. Assume that $C = s_c \rightarrow b_c$ subsumes $[s, c]$. Hence, there is a substitution θ such that $s_c \cdot \theta \subseteq s$ and $b_c \cdot \theta \in c$. To see which clause in $U(C)$ is captured by $[s, c]$ consider the partition π defined by the way terms in $s_c \cup \{b_c\}$ are unified by the substitution θ. More precisely, two distinct terms t, t' in $Terms([s_c, b_c])$ fall into the same class of π if and only if $t \cdot \theta = t' \cdot \theta$. Assume π has l classes. The proof argues that the clause $U_\pi(C)$ appears in $U(C)$ and that $[s, c]$ captures $U_\pi(C)$.

We observe that θ is a unifier for $A_\pi = \{A(t_1, \ldots, t_l) \mid t_1 \in \pi_1 \wedge \ldots \wedge t_l \in \pi_l\}$. Thus, an *mgu* σ_π exists. Therefore, $\theta = \sigma_\pi \cdot \hat{\theta}$ for some substitution $\hat{\theta}$. The transformation procedure rejects a partition π when any of the following conditions

holds. Either A_π is not unifiable (however, we have seen it is) or the representatives of two distinct classes are equal. The second condition does not hold because $\pi_i \cdot \sigma_\pi = \pi_j \cdot \sigma_\pi$ (for $i \neq j$) implies $\pi_i \cdot \theta = \pi_j \cdot \theta$, which is not true by construction.

Finally, we show that $[s, c]$ captures $U_\pi(C) = (s_t \xrightarrow{\neq} b_t)$ via $\hat{\theta}$. Notice that $s_c \cdot \sigma_\pi = s_t$ and $b_c \cdot \sigma_\pi = b_t$. We need to check (1) $s_t \cdot \hat{\theta} \subseteq s$, (2) $ineq(s_t \cup \{b_t\}) \cdot \hat{\theta} \subseteq ineq(s \cup c)$ and (3) $b_t \cdot \hat{\theta} \in c$. Condition (1) is easy: $s_t \cdot \hat{\theta} = s_c \cdot \sigma_\pi \cdot \hat{\theta} = s_c \cdot \theta \subseteq s$ by hypothesis. For (2), let t, t' be two different terms in $s_t \cup \{b_t\}$. It is sufficient to check that $t \cdot \hat{\theta}, t' \cdot \hat{\theta}$ are also different terms (i.e., $\hat{\theta}$ does not unify them). Let t_c, t'_c be the two terms in C such that $t_c \cdot \sigma_\pi = t$ and $t'_c \cdot \sigma_\pi = t'$. Since $t \neq t'$, it follows that t_c, t'_c belong to a different class of π (otherwise σ_π would have unified them). Therefore, by construction, $t_c \cdot \theta \neq t'_c \cdot \theta$. Equivalently, $t_c \cdot \sigma_\pi \cdot \hat{\theta} \neq t'_c \cdot \sigma_\pi \cdot \hat{\theta}$ and hence $t \cdot \hat{\theta} \neq t' \cdot \hat{\theta}$ as required. Condition (3) is like (1). ■

Lemma 22 *If $[s, c]$ is full w.r.t. some closed target expression T and $c \neq \emptyset$, then some clause of $U(T)$ must be captured by $[s, c]$.*

Proof. Fix any $b \in c$. Clearly, $T \models s \rightarrow b$ (since we have assumed $[s, c]$ full and hence correct and complete). Consider any minimal derivation graph G of $s \rightarrow b$ from T, which is guaranteed to exist by Theorem 2. Notice that all the participating atoms in G are in $Atoms_P(s \rightarrow b)$ since T is closed. Hence, they also appear in either s or c because $[s, c]$ is complete. Let $Pred(x)$ the set of atoms that have an edge ending at x in G. Let b' be an atom in G s.t. $Pred(b') \subseteq s$ and $b' \in c$. Such an atom must exist by definition of derivation graph. Hence, $Pred(b') \rightarrow b'$ is an instance of a clause in T and therefore it also subsumes $[s, c]$. By Lemma 21, we conclude that some clause in $U(T)$ is captured by $[s, c]$. ■

Lemma 23 *If $[s, c]$ captures some clause of $U(T)$, then $rhs(T, [s, c]) \neq \emptyset$.*

Proof. By assumption, there is a clause $s_c \rightarrow b_c$ in T and a substitution θ such that $s_c \cdot \theta \subseteq s$ and $b_c \cdot \theta \in c$. Clearly, $T \models s_c \rightarrow b_c \models s_c \cdot \theta \rightarrow b_c \cdot \theta$, therefore $b_c \cdot \theta \in rhs(T, [s, c])$, and hence $rhs(T, [s, c]) \neq \emptyset$ as required. ■

Corollary 24 *If $[s, c]$ is full w.r.t. T and $c \neq \emptyset$, then $rhs(T, [s, c]) \neq \emptyset$.* ■

5.2.5 Properties of minimized meta-clauses

This section includes properties of minimized meta-clauses as produced by the minimization procedure. Throughout the proof, we refer to the minimized meta-clause as $[s_x, c_x]$.

Lemma 28 shows that every minimized counterexample contains a syntactic variant of some clause in $U(T)$, if we ignore inequalities. This is an important property and it is responsible for one of the main improvements in the bounds.

Definition 28 A meta-clause $[s, c]$ is a positive counterexample for some target expression T and some hypothesis H if $T \models [s, c]$, $c \neq \emptyset$ and for all atoms $b \in c$, $H \not\models s \rightarrow b$.

Lemma 25 *Every minimized $[s_x, c_x]$ is full w.r.t. the target expression T.*

Proof. We proceed by induction on the updates of $[s_x, c_x]$ during computation of the minimization procedure. Our base case is the first version of the counterexample $[s_x, c_x]$ as produced by step 1 of the algorithm. This meta-clause is full, since it is the output of Cons-Closure which produces full meta-clauses by definition.

To see that the final meta-clause is correct it suffices to observe that every time the candidate meta-clause has been updated, the consequent part is computed as the output of rhs. Therefore, it must be correct.

To see that the final meta-clause is complete, we prove first that after generalizing a term the resulting counterexample remains complete. Let $[s_x, c_x]$ be the meta-clause before generalizing t and $[s'_x, c'_x]$ after. Let $\theta_t = \{x_t \mapsto t\}$. Then, $s'_x \cdot \theta_t = s_x$ and $c_x = c'_x \cdot \theta_t$, because x_t is a new variable that does not appear in $[s_x, c_x]$. By way of contradiction, suppose that some atom $b \in Atoms_P(s'_x \cup c'_x) \setminus s'_x$ such that $T \models s'_x \rightarrow b$ is not in c'_x. Notice that the substitution θ_t is non-unifying w.r.t. $s'_x \cup c'_x$, and therefore using properties 2 and 4 in Lemma 1 we conclude that

$b \cdot \theta_t \in Atoms_P(s_x \cup c_x) \setminus s_x$ and $b \cdot \theta_t \notin c_x$. Since $T \models s_x \rightarrow b \cdot \theta_t$, this contradicts our (implicit) induction hypothesis stating that $[s_x, c_x]$ is complete, since the atom $b \cdot \theta_t$ would be missing. Hence, any counterexample $[s_x, c_x]$ after step 2 is complete.

We show now that after dropping some term t the meta-clause still remains complete. Again, let $[s_x, c_x]$ be the meta-clause before removing t and $[s'_x, c'_x]$ after removing it. It is clear that $s'_x \subseteq s_x$ and $c'_x \subseteq c_x$ since both have been obtained by only removing atoms. By the induction hypothesis, the only atoms that could be missing are atoms in $c_x \setminus c'_x$ and $s_x \setminus s'_x$. Since for the closure of $[s'_x, c'_x]$ we only consider atoms in $Atoms_P(s'_x \cup c'_x)$ and these atoms do not contain t (all occurrences have been removed), the removed atoms cannot be missing because they all contain t. Therefore, after step 3 and as returned by the minimization procedure, the counterexample $[s_x, c_x]$ is complete. ■

Lemma 26 *At all times, $T \models H$ and all counterexamples given by the equivalence query oracle are positive, i.e., it is implied by target T but not by hypothesis H.*

Proof. We argue first that all meta-clauses in S in LEARN-CLOSED-HORN are correct. This is easy to see since every time S grows is either by adding a minimized counterexample (line 7 in LEARN-CLOSED-HORN), which by Lemma 25 is correct, or by replacing an existing meta-clause (line 6). Notice that the meta-clause that replaces the old one has RHS as its consequent, and hence it has to be correct. If all the meta-clauses in S are correct then, by definition, $T \models H$.

If $T \models H$ then every counterexample $A \rightarrow a$ given by the equivalence query oracle must be such that $T \models A \rightarrow a$ but $H \not\models A \rightarrow a$. ■

Lemma 27 *Every minimized $[s_x, c_x]$ is a positive counterexample w.r.t. target T and hypothesis H.*

Proof. Let $A \rightarrow a$ be the original counterexample obtained from the equivalence oracle. To prove that $[s_x, c_x]$ is a positive counterexample we need to prove that

$T \models [s_x, c_x]$, $c_x \neq \emptyset$ and for every $b \in c_x$ it holds that $H \not\models s_x \rightarrow b_x$. By Lemma 25, we know that $[s_x, c_x]$ is full, and hence correct so that $T \models [s_x, c_x]$. Moreover, $a \in c_x$ after step 1 so that $c_x \neq \emptyset$, since $T \models A \rightarrow a$ but $H \not\models A \rightarrow a$ so that a is not placed in the antecedent of the clause after executing ANT-CLOSURE but it is placed in c_x by CONS-CLOSURE. The meta-clause is further refined in steps 2 and 3 only if the consequent is non-empty. It remains to show that H does not imply any of the clauses in $[s_x, c_x]$.

The call to the procedure ANT-CLOSURE guarantees that every atom implied by H is placed into the antecedent s_x, leaving no space for any atom implied by H to be put into the consequent c_x by CONS-CLOSURE. Thus, after step 1 of the minimization procedure, $[s_x, c_x]$ is still a positive counterexample.

Next, we see that after generalizing some functional term t, the meta-clause still remains a positive counterexample. Let $[s_x, c_x]$ be the meta-clause before generalizing t, and $[s'_x, c'_x]$ after. Assume $[s_x, c_x]$ is a positive counterexample. Let θ_t be the substitution $\{x_t \mapsto t\}$. As in Lemma 25, $s'_x \cdot \theta_t = s_x$ and $c'_x \cdot \theta_t = c_x$. Suppose by way of contradiction that $H \models s'_x \rightarrow b'$, for some $b' \in c'_x$. Then, $H \models s'_x \cdot \theta_t \rightarrow b' \cdot \theta_t$, which contradicts the fact that $[s_x, c_x]$ was a positive example since $b' \cdot \theta_t \in c_x$.

Finally, we show that after dropping some term t the meta-clause still remains a positive counterexample. As before, let $[s_x, c_x]$ be the meta-clause before removing some of its atoms, and $[s'_x, c'_x]$ after. Assume $[s_x, c_x]$ is a positive counterexample, hence, $H \not\models s_x \rightarrow b$ for all $b \in c_x$. Clearly, $H \not\models s'_x \rightarrow b$ for all $b \in c'_x$ because $s'_x \subseteq s_x$ and $c'_x \subseteq c_x$, and $[s'_x, c'_x]$ is positive. ■

Lemma 28 *If a minimized $[s_x, c_x]$ captures some clause $s_t \xrightarrow{\neq} b_t$ of $U(T)$, then it must be via some substitution θ such that θ is a variable renaming, i.e., θ maps distinct variables of s_t into distinct variables of s_x only.*

Proof. $[s_x, c_x]$ captures $s_t \xrightarrow{\neq} b_t$, hence there must exist a substitution θ from variables in $s_t \cup \{b_t\}$ into terms in $s_x \cup c_x$ such that $s_t \cdot \theta \subseteq s_x$, $ineq(s_t \cup \{b_t\}) \cdot \theta \subseteq ineq(s_x \cup c_x)$ and $b_t \cdot \theta \in c_x$. We show that θ must be a variable renaming.

By way of contradiction, suppose that θ maps some variable v of $s_t \cup \{b_t\}$ into a functional term t of $s_x \cup c_x$ (i.e. $v \cdot \theta = t$). Consider the generalization of the term t in step 2 of the minimization procedure. We see that the term t should have been generalized and substituted by the new variable x_t.

Suppose, then, that $[s_x, c_x]$ is the meta-clause just before attempting to generalize t and that $[s'_x, c'_x]$ is the meta-clause obtained after generalizing the term t to the fresh variable x_t. Consider the substitution $\theta' = \theta \setminus \{v \mapsto t\} \cup \{v \mapsto x_t\}$. The substitution θ' behaves like θ on all terms except for variable v. We see that $[s'_x, c'_x]$ captures $s_t \xrightarrow{\neq} b_t$ via θ' and hence $\textsc{rhs}(s'_x, c'_x) \neq \emptyset$ (Lemma 23). Therefore t must be generalized to the variable x_t.

To see that $[s'_x, c'_x]$ captures $s_t \xrightarrow{\neq} b_t$ via θ' we need to show (1) $s_t \cdot \theta' \subseteq s'_x$, (2) $b_t \cdot \theta' \in c'_x$ and (3) $\mathit{ineq}(s_t \cup \{b_t\}) \cdot \theta' \subseteq \mathit{ineq}(s'_x \cup c'_x)$. For (1), consider any atom b of s_t. We observe the following: after substitution θ': $b(..v..) \Rightarrow b(..x_t..)$, and after substitution θ and generalizing t: $b(..v..) \Rightarrow b(..t..) \Rightarrow b(..x_t..)$. The part of the "dots" in the previous expressions is identical for both lines, since θ and θ' behave equally for terms different than v. Moreover, the fact that θ does not unify terms in $s_t \cup \{b_t\}$ assures that the rest of terms differ from t and x_t after applying θ or θ'. Therefore, we get that $b \cdot \theta' \in s'_x$ iff $b \cdot \theta \in s_x$ and since $s_t \cdot \theta \subseteq s_x$, Property (1) follows. Property (2) is identical to Property (1). For (3), let t, t' be two distinct terms of $s_t \cup \{b_t\}$. We have to show that $t \cdot \theta'$ and $t' \cdot \theta'$ are two different terms of $s'_x \cup c'_x$ and therefore their inequality appears in $\mathit{ineq}(s'_x \cup c'_x)$. It is easy to see that they are terms of $s'_x \cup c'_x$ since by previous properties $(s_t \cup \{b_t\}) \cdot \theta' \subseteq (s'_x \cup c'_x)$. Now, let θ_t be the substitution $\{x_t \to t\}$ and notice that $\theta = \theta' \cdot \theta_t$. Since θ does not unify terms in $s_t \cup \{b_t\}$, then neither of θ' and θ_t do. Therefore, $t \cdot \theta' \neq t' \cdot \theta'$.

■

5.2.6 On the number of terms in minimized examples

Lemma 29 *Let $[s_x, c_x]$ be a minimized meta-clause. Let $s_t \xrightarrow{\neq} b_t$ be a clause of $U(T)$ captured by $[s_x, c_x]$. Then, $|Terms(s_x \cup \{c_x\})| = |Terms(s_t \cup \{b_t\})|$.*

Proof. Let n_x and n_t be the number of distinct terms appearing in $[s_x, c_x]$ and $s_t \rightarrow b_t$, respectively. Subterms should also be counted. The meta-clause $[s_x, c_x]$ captures $s_t \xrightarrow{\neq} b_t$. Therefore there is a substitution θ satisfying $ineq(s_t \cup \{b_t\}) \cdot \theta \subseteq ineq(s_x \cup c_x)$. Thus, different variables in $s_t \rightarrow b_t$ are mapped into different terms of $s_x \cup c_x$ by θ. By Lemma 28, we know also that every variable of s_t, b_t is mapped into a variable of s_x, c_x. Therefore, θ maps distinct variables of s_t, b_t into distinct variables of s_x, c_x. Therefore, the number of terms in s_t, b_t equals the number of terms in $(s_t \cup \{b_t\}) \cdot \theta$, since there has only been a non-unifying renaming of variables. Also, $s_t \cdot \theta \subseteq s_x$ and $b_t \cdot \theta \in c_x$. We have to check that the remaining atoms in $(s_x \setminus s_t \cdot \theta) \cup (c_x \setminus b_t \cdot \theta)$ do not include any term not appearing in $(s_t \cup \{b_t\}) \cdot \theta$.

Suppose there is an atom $l \in (s_x \setminus s_t \cdot \theta) \cup (c_x \setminus b_t \cdot \theta)$ containing some term, say t, not appearing in $(s_t \cup \{b_t\}) \cdot \theta$. Consider when in step 3 of the minimization procedure the term t was checked as a candidate to be removed. Let $[s'_x, c'_x]$ be the clause obtained after the removal of the atoms containing t. Then, $s_t \cdot \theta \subseteq s'_x$ and $b_t \cdot \theta \in c'_x$ because all the atoms in $(s_t \cup \{b_t\}) \cdot \theta$ do not contain t. Moreover, $ineq(s_t \cup \{b_t\}) \cdot \theta \subseteq ineq(s'_x \cup c'_x)$. To see this, take any two terms $t \neq t'$ from $s_t \rightarrow b_t$. The terms $t \cdot \theta$ and $t' \cdot \theta$ appear in $s'_x \cup c'_x$ because they contain terms in $(s_t \cup b_t) \cdot \theta$ only (so they are not removed). Further, since $t \cdot \theta \neq t' \cdot \theta$ in $s_x \cup c_x$ and $\{t \cdot \theta, t' \cdot \theta\} \subseteq (s'_x \cup c'_x) \subseteq (s_x \cup c_x)$ we conclude that $t \cdot \theta \neq t' \cdot \theta$ in $s'_x \cup c'_x$. Thus, $[s'_x, c'_x]$ still captures $s_t \xrightarrow{\neq} b_t$. And therefore, $\text{RHS}(s'_x, c'_x) \neq \emptyset$ and such a term t cannot exist. We conclude that $n_t = n_x$. ■

Corollary 30 *Let t be an upper bound on the number of distinct terms in any target clause. Then, the number of terms of a minimized counterexample is at most t.* ■

Lemma 31 *Let $[s, c]$ be a meta-clause covering a fully inequated clause $s_t \xrightarrow{\neq} b_t$. Then, $|Terms(s_t \cup \{b_t\})| \leq |Terms(s \cup c)|$*

Proof. Since $[s, c]$ covers the clause $s_t \xrightarrow{\neq} b_t$, there is a θ s.t. $ineq(s_t \cup \{b_t\}) \cdot \theta \subseteq ineq(s \cup c)$. Therefore, any two distinct terms of $s_t \cup \{b_t\}$ appear as distinct terms in $s \cup c$. And therefore, $[s, c]$ has at least as many terms as $s_t \rightarrow b_t$. ■

Corollary 32 *Let $s_t \xrightarrow{\neq} b_t$ be a fully inequated clause in $U(T)$. Let $[s_x, c_x]$ be a minimized counterexample capturing $s_t \xrightarrow{\neq} b_t$, and let $[s_i, c_i]$ be any other meta-clause covering $s_t \xrightarrow{\neq} b_t$. Then, $|Terms(s_x \cup c_x)| \leq |Terms(s_i \cup c_i)|$.* ■

5.2.7 Properties of pairings

Lemma 33 *Let $[s_x, c_x]$ and $[s_i, c_i]$ be two full meta-clauses w.r.t. some closed target expression T. Let $[s, c] \in$ Basic-Pairings$([s_x, c_x], [s_i, c_i])$. Then, $[s, \text{RHS}([s, c])]$ is full w.r.t. T.*

Proof. To see that $[s, \text{RHS}([s, c])]$ is full w.r.t. T, it is sufficient to show that $[s, c]$ is complete — RHS takes care of the resulting meta-clause being correct. Suppose $T \models s \rightarrow b$ for some $b \in Atoms_P(s \cup c) \setminus s$. Since $s = lgg_{|\sigma}(s_x, s_i) \subseteq lgg(s_x, s_i)$, we know that there exist θ_x and θ_i such that $s \cdot \theta_x \subseteq s_x$ and $s \cdot \theta_i \subseteq s_i$. $T \models s \rightarrow b$ implies both $T \models s \cdot \theta_x \rightarrow b \cdot \theta_x$ and $T \models s \cdot \theta_i \rightarrow b \cdot \theta_i$. Let $b_x = b \cdot \theta_x$ and $b_i = b \cdot \theta_i$ so that $T \models s_x \rightarrow b_x$ and $T \models s_i \rightarrow b_i$. By assumption, $[s_x, c_x]$ and $[s_i, c_i]$ are full, and therefore $b_x \in s_x \cup c_x$ and $b_i \in s_i \cup c_i$ because $b_x \in Atoms_P(s_x \cup c_x)$ and $b_i \in Atoms_P(s_i \cup c_i)$ (remember that $b \in Atoms_P(s \cup c)$). Also, since the same *lgg* table is used for all $lgg(\cdot, \cdot)$ we know that $b = lgg(b_x, b_i)$. Therefore b must appear in one of $lgg(s_x, s_i), lgg(s_x, c_i), lgg(c_x, s_i)$ or $lgg(c_x, c_i)$. But $b \notin lgg(s_x, s_i)$ since $b \notin s$ by assumption.

Note that all terms and subterms in b appear in $s \cup c$, because $b \in Atoms_P(s \cup c)$. Let σ be the basic matching inducing $[s, c]$. We know that σ is basic and hence legal, and therefore it contains all subterms of terms appearing in $s \cup c$. Thus, by

restricting any of the $lgg(\cdot,\cdot)$ to $lgg_{|_\sigma}(\cdot,\cdot)$, we do not get rid of b, since it is built up from terms that appear in $s \cup c$ and hence in σ. Therefore, $b \in lgg_{|_\sigma}(s_x, c_i) \cup lgg_{|_\sigma}(c_x, s_i) \cup lgg_{|_\sigma}(c_x, c_i) = c$ as required. ■

Lemma 34 *Let* $[s,c] \in$ *Basic-Pairings*$([s_x,c_x],[s_i,c_i])$. *Then,* $|s| \leq min(|s_i|, |s_x|)$ *and* $|s \cup c| \leq min(|s_i \cup c_i|, |s_x \cup c_x|)$.

Proof. It is sufficient to observe that in s there is at most one copy of every atom in s_i. This is true since the matching used to include atoms in s is 1 to 1 and therefore a term can only be combined with a unique term and no duplication of atoms occurs. The same idea applies to s_x and the second inequality. ■

Lemma 35 *Let* $[s_1,c_1]$ *and* $[s_2,c_2]$ *be two full meta-clauses w.r.t. some closed Horn expression* T, *let* $[s,c]$ *be any legal pairing between them, and let* $s_t \xrightarrow{\neq} b_t \in U(T)$. *The following holds:*

1. *If* $[s,c]$ *covers* $s_t \xrightarrow{\neq} b_t$, *both* $[s_1,c_1]$ *and* $[s_2,c_2]$ *cover* $s_t \xrightarrow{\neq} b_t$.
2. *If* $[s,c]$ *captures* $s_t \xrightarrow{\neq} b_t$, *at least one of* $[s_1,c_1]$ *or* $[s_2,c_2]$ *captures* $s_t \xrightarrow{\neq} b_t$.

Proof. Condition 1: by assumption, $s_t \xrightarrow{\neq} b_t$ is covered by $[s,c]$, i.e., there is a θ such that $s_t \cdot \theta \subseteq s$, $ineq(s_t \cup \{b_t\}) \cdot \theta \subseteq ineq(s \cup c)$ and $b_t \cdot \theta \in Atoms_P(s \cup c)$. This implies that if t, t' are two distinct terms of $s_t \cup \{b_t\}$, then $t \cdot \theta$ and $t' \cdot \theta$ are distinct terms appearing in $s \cup c$. Let σ be the 1-1 legal matching inducing the pairing. The antecedent s is defined to be $lgg_{|_\sigma}(s_1, s_2)$, and therefore there exist substitutions θ_1 and θ_2 such that $s \cdot \theta_1 \subseteq s_1$ and $s \cdot \theta_2 \subseteq s_2$. We claim that $[s_1,c_1]$ and $[s_2,c_2]$ cover $s_t \xrightarrow{\neq} b_t$ via $\theta \cdot \theta_1$ and $\theta \cdot \theta_2$, respectively. We prove this for $[s_1,c_1]$ only, the proof for $[s_2,c_2]$ is identical. Notice that $s_t \cdot \theta \subseteq s$, and therefore $s_t \cdot \theta \cdot \theta_1 \subseteq s \cdot \theta_1$. Since $s \cdot \theta_1 \subseteq s_1$, we obtain $s_t \cdot \theta \cdot \theta_1 \subseteq s_1$. We show now that $ineq(s_t \cup \{b_t\}) \cdot \theta \cdot \theta_1 \subseteq ineq(s_1 \cup c_1)$. Observe that all top-level terms appearing in $s \cup c$ also appear as one entry of the matching σ, because otherwise they could not have survived the restriction imposed by σ. Further, since σ is legal, all subterms of

terms of $s \cup c$ also appear as an entry in σ. Let t, t' be two distinct terms appearing in $s_t \cup \{b_t\}$. Since $(s_t \cup \{b_t\}) \cdot \theta \subseteq s \cup c$ and σ includes all terms appearing in $s \cup c$, the distinct terms $t \cdot \theta$ and $t' \cdot \theta$ appear as the *lgg* of distinct entries in σ. These entries have the form `[`$t \cdot \theta \cdot \theta_1$ `-` $t \cdot \theta \cdot \theta_2$ `=>` $t \cdot \theta$`]`, since $lgg(t \cdot \theta \cdot \theta_1, t \cdot \theta \cdot \theta_2) = t \cdot \theta$. Since σ is 1-1, we know that $t \cdot \theta \cdot \theta_1 \neq t' \cdot \theta \cdot \theta_1$. Finally, we need to show that $b_t \cdot \theta \cdot \theta_1 \in Atoms_P(s_1 \cup c_1)$. Notice that $s \cdot \theta_1 \subseteq s_1$ and $c \cdot \theta_1 \subseteq (s_1 \cup c_1)$. Therefore, $s_t \cup \{b_t\} \cdot \theta \subseteq s \cup c$ implies $s_t \cup \{b_t\} \cdot \theta \cdot \theta_1 \subseteq (s \cup c) \cdot \theta_1 \subseteq s_1 \cup c_1$. Thus, $b_t \cdot \theta \cdot \theta_1 \in Atoms_P(s_1 \cup c_1)$ as required.

Condition 2: by hypothesis, $b_t \cdot \theta \in c$ and c is defined to be $lgg_{|_\sigma}(s_1, c_2) \cup lgg_{|_\sigma}(c_1, s_2) \cup lgg_{|_\sigma}(c_1, c_2)$. Observe that all these *lgg*s share the same table, so the same pairs of terms are mapped into the same expressions. Observe also that the substitutions θ_1 and θ_2 are defined according to this table, so that if any atom $l \in lgg_{|_\sigma}(c_1, \cdot)$, then $l \cdot \theta_1 \in c_1$. Equivalently, if $l \in lgg_{|_\sigma}(\cdot, c_2)$, then $l \cdot \theta_2 \in c_2$. Therefore we get that if $b_t \cdot \theta \in lgg_{|_\sigma}(c_1, \cdot)$, then $b_t \cdot \theta \cdot \theta_1 \in c_1$ and if $b_t \cdot \theta \in lgg_{|_\sigma}(\cdot, c_2)$, then $b_t \cdot \theta \cdot \theta_2 \in c_2$. Now, observe that in any of the three possibilities for c, one of c_1 or c_2 is included in the $lgg_{|_\sigma}$. Thus it is the case that either $b_t \cdot \theta \cdot \theta_1 \in c_1$ or $b_t \cdot \theta \cdot \theta_2 \in c_2$. Since both $[s_1, c_1]$ and $[s_2, c_2]$ cover $s_t \xrightarrow{\neq} b_t$, one of $[s_1, c_1]$ or $[s_2, c_2]$ captures $s_t \xrightarrow{\neq} b_t$. ■

It is crucial for Lemma 35 that the pairing involved is *legal*. It is indeed possible for a *non-legal* pairing to capture some clause that is not even covered by some of its originating meta-clauses, as the next example illustrates.

Example 12 In this example we present two meta-clauses $[s_1, c_1]$ and $[s_2, c_2]$, a non-legal matching σ and a clause $s_t \xrightarrow{\neq} b_t$ such that the non-legal pairing induced by σ captures $s_t \xrightarrow{\neq} b_t$ but none of $[s_1, c_1]$ and $[s_2, c_2]$ do.

- $[s_1, c_1] = [p(fffa, gffa) \rightarrow q(fa)]$ with terms $\{a, fa, ffa, gffa\}$
  $ineq(s_1) = (a \neq fa \neq ffa \neq gffa)$.
- $[s_2, c_2] = [p(fb, gffc) \rightarrow q(b)]$ with terms $\{b, c, fb, fc, ffc, gffc\}$.

- The matching σ is `[a - c => X]`
  `[fa - b => Y]`
  `[ffa - fb => fY]`
  `[gffa - gffc => gffX]`

- $[s, c] = [p(fY, gffX) \rightarrow q(Y)]$.

- $\underbrace{(x \neq fx \neq ffx \neq gffx \neq y \neq fy)}_{ineq(s_t)}, \underbrace{p(fy, gffx)}_{s_t} \rightarrow \underbrace{q(y)}_{b_t}$.

- $\theta = \{x \mapsto X, y \mapsto Y\}$.

- $\theta_1 = \{X \mapsto a, Y \mapsto fa\}$.

- $\theta \cdot \theta_1 = \{x \mapsto a, y \mapsto fa\}$.

The meta-clause $[s, c]$ captures $s_t \xrightarrow{\neq} b_t$ via $\theta = \{x \mapsto X, y \mapsto Y\}$. But $[s_1, c_1]$ does not cover $s_t \xrightarrow{\neq} b_t$ because the condition $ineq(s_t) \cdot \theta \cdot \theta_1 \subseteq ineq(s_1)$ fails to hold (the terms that violate inequalities are highlighted by the boxes below, e.g., fx and y are both mapped to fa by $\theta \cdot \theta_1$):

$$\underbrace{(a \neq \boxed{\text{fa}} \neq \boxed{\text{ffa}} \neq gffa \neq \boxed{\text{fa}} \neq \boxed{\text{ffa}})}_{(x \neq fx \neq ffx \neq gffx \neq y \neq fy) \cdot \theta \cdot \theta_1} \not\subseteq \underbrace{(a \neq fa \neq ffa \neq gffa)}_{ineq(s_1)}$$

Corollary 36 *Let $[s_1, c_1], [s_2, c_2], [s_3, c_3], \ldots, [s_k, c_k], \ldots$ be a sequence of full meta-clauses such that every meta-clause $[s_{i+1}, c_{i+1}]$ is a legal pairing between the previous meta-clause $[s_i, c_i]$ in the sequence and some other full meta-clause $[s'_i, c'_i]$, for $i \geq 1$. Suppose some $[s_k, c_k]$ in the sequence covers a clause $s_t \xrightarrow{\neq} b_t$. Then, all previous $[s_i, c_i]$ in the sequence (where $i < k$), must cover the clause $s_t \xrightarrow{\neq} b_t$, too.* ■

5.2.8 Properties of the sequence S

Corollary 37 *Every meta-clause $[s_i, c_i]$ appearing in the sequence S is full w.r.t. the target expression T.*

Proof. The sequence S is constructed by appending minimized counterexamples or by refining existing elements with a pairing with another minimized counterexample. Lemma 25 guarantees that all minimized counterexamples are full and, by Lemma 33, any basic pairing between full meta-clauses is also full. ■

Lemma 38 *Let S be the sequence $[[s_1, c_1], [s_2, c_2], \ldots, [s_k, c_k]]$. If a minimized counterexample $[s_x, c_x]$ is produced such that it captures some clause $s_t \xrightarrow{\neq} b_t$ in $U(T)$ covered by some $[s_i, c_i]$ of S, then some meta-clause $[s_j, c_j]$ is replaced by a basic pairing of $[s_x, c_x]$ and $[s_j, c_j]$, where $j \leq i$.*

Proof. We show that if no element $[s_j, c_j]$ where $j < i$ is replaced, then $[s_i, c_i]$ itself must be replaced. We have to prove that there exists a basic pairing $[s, c]$ in $\mathit{Basic\text{-}Pairings}([s_x, c_x], [s_i, c_i])$ satisfying the replacement conditions: $\textsc{rhs}([s, c]) \neq \emptyset$ and $\mathit{WSize}([s, c]) < \mathit{WSize}([s_i, c_i])$.

We have assumed that there is some clause $s_t \xrightarrow{\neq} b_t \in U(T)$ captured by $[s_x, c_x]$ and covered by $[s_i, c_i]$. Let θ'_x be the substitution showing that $s_t \xrightarrow{\neq} b_t$ is captured by $[s_x, c_x]$ and θ'_i the substitution showing that $s_t \xrightarrow{\neq} b_t$ is covered by $[s_i, c_i]$. Thus:

- $s_t \cdot \theta'_x \subseteq s_x$
- $\mathit{ineq}(s_t \cup \{b_t\}) \cdot \theta'_x \subseteq \mathit{ineq}(s_x \cup c_x)$
- $b_t \cdot \theta'_x \in c_x$
- $b_t \cdot \theta'_x \in \mathit{Atoms}_P(s_x \cup c_x)$
- $s_t \cdot \theta'_i \subseteq s_i$
- $\mathit{ineq}(s_t \cup \{b_t\}) \cdot \theta'_i \subseteq \mathit{ineq}(s_i \cup c_i)$
- $b_t \cdot \theta'_i \in \mathit{Atoms}_P(s_i \cup c_i)$

We construct a matching σ that includes all entries

$$[t \cdot \theta'_x \ \texttt{-} \ t \cdot \theta'_i \ \texttt{=>} \ \mathit{lgg}(t \cdot \theta'_x, t \cdot \theta'_i)]$$

such that t is a term appearing in $s_t \cup \{b_t\}$ (one entry for every distinct term).

Example 13 Consider the following:

- $s_t = \{p(g(c), x, f(y), z)\}$.
  With terms $c, g(c), x, y, f(y)$ and z.

- $s_x = \{p(g(c), x', f(y'), z), p(g(c), g(c), f(y'), c)\}$.
  With terms $c, g(c), x', y', f(y')$ and z.

- $s_i = \{p(g(c), f(1), f(f(2)), z)\}$.
  With terms $c, g(c), 1, f(1), 2, f(2), f(f(2))$ and z.

- The substitution $\theta'_x = \{x \mapsto x', y \mapsto y', z \mapsto z\}$, which is a variable renaming.

- The substitution $\theta'_i = \{x \mapsto f(1), y \mapsto f(2), z \mapsto z\}$.

- The $lgg(s_x, s_i)$ is $\{p(g(c), X, f(Y), z), p(g(c), Z, f(Y), V)\}$ and it produces the following *lgg* table.

```
[c - c => c]                  [g(c) - g(c) => g(c)]
[x' - f(1) => X]              [y' - f(2) => Y]
[f(y') - f(f(2)) => f(Y)]     [z - z => z]
[g(c) - f(1) => Z]            [c - z => V]
```

- The extended matching σ is

$$
\begin{aligned}
c &\Rightarrow \texttt{[c - c => c]} \\
g(c) &\Rightarrow \texttt{[g(c) - g(c) => g(c)])} \\
x &\Rightarrow \texttt{[x' - f(1) => X]} \\
y &\Rightarrow \texttt{[y' - f(2) => Y]} \\
f(y) &\Rightarrow \texttt{[f(y') - f(f(2)) => f(Y)]} \\
z &\Rightarrow \texttt{[z - z => z]}
\end{aligned}
$$

- The pairing induced by σ is $lgg_{|_\sigma}(s_x, s_i) = \{p(g(c), X, f(Y), z)\}$.

Claim 39 *The matching σ as described above is 1-1 and the number of entries equals the minimum of the number of distinct terms in $s_x \cup c_x$ and $s_i \cup c_i$.*

Proof. All the entries of σ have the form $[t \cdot \theta'_x \texttt{ - } t \cdot \theta'_i \texttt{ => } lgg(t \cdot \theta'_x, t \cdot \theta'_i)]$. For σ to be 1-1 it is sufficient to see that there are no two terms t, t' of $s_t \cup \{b_t\}$ generating the following entries in σ

$$[t \cdot \theta'_x \texttt{ - } t \cdot \theta'_i \texttt{ => } lgg(t \cdot \theta'_x, t \cdot \theta'_i)]$$
$$[t' \cdot \theta'_x \texttt{ - } t' \cdot \theta'_i \texttt{ => } lgg(t' \cdot \theta'_x, t \cdot \theta'_i)]$$

such that $t \cdot \theta'_x = t' \cdot \theta'_x$ or $t \cdot \theta'_i = t' \cdot \theta'_i$. But this is clear since $[s_x, c_x]$ and $[s_i, c_i]$ are covering $s_t \xrightarrow{\neq} b_t$ via θ'_x and θ'_i, respectively. Therefore $ineq(s_t \cup \{b_t\}) \cdot \theta'_x \subseteq ineq(s_x \cup c_x)$ and $ineq(s_t \cup \{b_t\}) \cdot \theta'_i \subseteq ineq(s_i \cup c_i)$. And therefore $t \cdot \theta'_x$ and $t' \cdot \theta'_x$ appear as different terms in $s_x \cup c_x$. Also, $t \cdot \theta'_i$ and $t' \cdot \theta'_i$ appear as different terms in $s_i \cup c_i$. Thus σ is 1-1.

By construction, the number of entries equals the number of distinct terms in $s_t \cup \{b_t\}$, that by Lemma 29 is the number of distinct terms in $s_x \cup c_x$. And by Lemma 31, $[s_i, c_i]$ contains at least as many terms as $s_t \cup \{b_t\}$. Therefore, the number of entries in σ coincides with the minimum of the number of distinct terms in $s_x \cup c_x$ and $s_i \cup c_i$. ■

Claim 40 *The matching σ is legal.*

Proof. A matching is legal if the subterms of any term appearing as the *lgg* of the matching also appear in some other entries of the matching. We prove it by induction on the structure of the terms. We prove that if t is a term in $s_t \cup \{b_t\}$, then the term $lgg(t \cdot \theta'_x, t \cdot \theta'_i)$ and all its subterms appear in σ's extension.

Base case. When $t = a$, with a being some constant. The entry in σ for it is `[a - a => a]`, since $a \cdot \theta = a$, for any substitution θ if a is a constant and $lgg(a, a) = a$. Trivially, all of a's subterms appear in σ.

Base case. When $t = v$, where v is any variable in $s_t \cup \{b_t\}$. The entry for it in σ is $[v \cdot \theta'_x \texttt{ - } v \cdot \theta'_i \texttt{ => } lgg(v \cdot \theta'_x, v \cdot \theta'_i)]$. Since $[s_x, c_x]$ is minimized, Lemma 28

guarantees that $v \cdot \theta'_x$ is a variable. Therefore, $lgg(v \cdot \theta'_x, v \cdot \theta'_i)$ must be a variable, regardless of what $v \cdot \theta'_i$ is. Trivially, all of its subterms appear in σ.

Step case. When $t = f(t_1, .., t_l)$, where f is a function symbol of arity l and $t_1, .., t_l$ its arguments. The entry for it in σ is

$$[f(t_1, .., t_l) \cdot \theta'_x \texttt{ - } f(t_1, .., t_l) \cdot \theta'_i \texttt{ => } \underbrace{lgg(f(t_1, .., t_l) \cdot \theta'_x, f(t_1, .., t_l) \cdot \theta'_x)}_{f(lgg(t_1 \cdot \theta'_x, t_1 \cdot \theta'_i), .., lgg(t_l \cdot \theta'_x, t_l \cdot \theta'_i))}]$$

The entries $[t_j \cdot \theta'_x \texttt{ - } t_j \cdot \theta'_i \texttt{ => } lgg(t_j \cdot \theta'_x, t_j \cdot \theta'_x)]$, with $1 \leq j \leq l$, are also included in σ, since all t_j are terms of $s_t \cup \{b_t\}$. By the induction hypothesis, all the subterms of every $lgg(t_j \cdot \theta'_x, t_j \cdot \theta'_x)$ are included in σ, and therefore, all the subterms of $lgg(f(t_1, .., t_l) \cdot \theta'_x, f(t_1, .., t_l) \cdot \theta'_x)$ are also included in σ. ■

Claim 41 *The matching σ is basic.*

Proof. A basic matching is defined only for two meta-clauses $[s_x, c_x]$ and $[s_i, c_i]$ such that the number of terms in $s_x \cup c_x$ is less or equal than the number of terms in $s_i \cup c_i$. Corollary 32 shows that this is indeed the case. The previous claims prove that σ is 1-1 and legal. It is only left to see that it is basic: if entry $f(t_1, .., t_n) - t$ is in σ, then $t = f(r_1, .., r_n)$ and $t_l - r_l \in \sigma$ for all $l = 1, .., n$.

Suppose, then, that $f(t_1, .., t_n) - t$ is in σ. By construction of σ all entries are of the form

$$[\hat{t} \cdot \theta'_x - \hat{t} \cdot \theta'_i],$$

where $\hat{t}$ is a term in $s_t \cup \{b_t\}$. Thus, assume $\hat{t} \cdot \theta'_x = f(t_1, .., t_n)$ and $\hat{t} \cdot \theta'_i = t$. We also know that θ'_x is a variable renaming, therefore, the term $\hat{t} \cdot \theta'_x$ is a variant of $\hat{t}$. Therefore, the terms $f(t_1, .., t_n)$ and $\hat{t}$ are variants. That is, $\hat{t}$ itself has the form $f(t'_1, .., t'_n)$, where every t'_j is a variant of t_j and $t'_j \cdot \theta'_x = t_j$, where $j = 1, .., n$. Therefore, $t = \hat{t} \cdot \theta'_i = f(r_1 = t'_1 \cdot \theta'_i, .., r_n = t'_n \cdot \theta'_i)$ as required. We have seen that $t_j = t'_j \cdot \theta'_x$ and $r_j = t'_j \cdot \theta'_i$. By construction, σ includes the entries $t_j - r_j$, for any $j = 1, .., n$ and our claim holds. ■

The claims above show that the matching σ is a good matching in the sense that it is one of the matchings constructed by the algorithm. The next claim shows

that $[s, c] = \text{PAIRING}(\sigma, [s_x, c_x], [s_i, c_i])$ is considered as a candidate for replacement in the learning algorithm LEARN-CLOSED-HORN.

Claim 42 $\text{PAIRING}(\sigma, [s_x, c_x], [s_i, c_i]) \in \textit{Basic-Pairings}([s_x, c_x], [s_i, c_i])$.

Proof. It is sufficient to observe that σ has been constructed precisely using the *lgg* of terms in $s_x \cup c_x$ and $s_i \cup c_i$, and it therefore agrees with the *lgg* table produced by the computation $lgg(s_x \cup c_x, s_i \cup c_i)$. ■

It is left to show that both conditions for replacement in the algorithm hold. The following two claims show that this is indeed the case.

Claim 43 $\text{RHS}([s, c]) \neq \emptyset$.

Proof. Let θ_x and θ_i be defined as follows. An entry in σ `[`$t \cdot \theta'_x$ `-` $t \cdot \theta'_i$ `=>` $lgg(t \cdot \theta'_x, t \cdot \theta'_i)$`]` such that $lgg(t \cdot \theta'_x, t \cdot \theta'_i)$ is a variable generates the mapping $lgg(t \cdot \theta'_x, t \cdot \theta'_i) \mapsto t \cdot \theta'_x$ in θ_x and $lgg(t \cdot \theta'_x, t \cdot \theta'_i) \mapsto t \cdot \theta'_i$ in θ_i. That is, $\theta_x = \{lgg(t \cdot \theta'_x, t \cdot \theta'_i) \mapsto t \cdot \theta'_x\}$ and $\theta_i = \{lgg(t \cdot \theta'_x, t \cdot \theta'_i) \mapsto t \cdot \theta'_i\}$, whenever $lgg(t \cdot \theta'_x, t \cdot \theta'_i)$ is a variable and t is a term in $s_t \cup \{b_t\}$.

In our example, $\theta_x = \{X \mapsto x', Y \mapsto y', z \mapsto z\}$ and $\theta_i = \{X \mapsto f(1), Y \mapsto f(2), z \mapsto z\}$. Next, we show that $s \cdot \theta_x \subseteq s_x$ and $s \cdot \theta_i \subseteq s_i$:

- $s \cdot \theta_x \subseteq s_x$. Let l be an atom in s, l has been obtained by taking the *lgg* of two atoms l_x and l_i in s_x and s_i, respectively. That is, $l = lgg(l_x, l_i)$. Moreover, l only contains terms in the extension of σ, otherwise it would have been removed when restricting the *lgg*. The substitution θ_x is such that $l \cdot \theta_x = l_x \in s_x$ because it "undoes" what the *lgg* does for the atoms with terms in σ.

- $s \cdot \theta_i \subseteq s_i$. Similar to previous.

Let θ be the substitution that maps all variables in $s_t \cup \{b_t\}$ to their corresponding expression assigned in the extension of σ. That is, θ maps any variable v of $s_t \cup \{b_t\}$ to the term $lgg(v \cdot \theta'_x, v \cdot \theta'_i)$. In our example, $\theta = \{x \mapsto X, y \mapsto Y, z \mapsto z\}$.

The strategy for the remainder of the proof consists in showing that $s_t \xrightarrow{\neq} b_t$ is captured by $[s, c]$ via θ. Applying Lemma 23 we then conclude that $\textsc{rhs}([s, c]) \neq \emptyset$.

Finally, the following properties show that $s_t \xrightarrow{\neq} b_t$ is captured by $[s, c]$ via θ:

– $\theta \cdot \theta_x = \theta'_x$**:** Let v be a variable in $s_t \cup \{b_t\}$. The substitution θ maps v into $lgg(v \cdot \theta'_x, v \cdot \theta'_i)$. This is a variable, say V, since we know θ'_x is a variable renaming. The substitution θ_x contains the mapping

$$\underbrace{lgg(v \cdot \theta'_x, v \cdot \theta'_i)}_{V} \mapsto v \cdot \theta'_x.$$

And v is mapped into $v \cdot \theta'_x$ by $\theta \cdot \theta_x$.

In our example: $\theta'_x = \{x \mapsto x', y \mapsto y', z \mapsto z\}$, and

$$\theta \cdot \theta_x = \{x \mapsto X, y \mapsto Y, z \mapsto z\} \cdot \{X \mapsto x', Y \mapsto y', z \mapsto z\}.$$

– $\theta \cdot \theta_i = \theta'_i$**:** As in previous property.

– $s_t \cdot \theta \subseteq s = lgg_{|\sigma}(s_x, s_i)$**:** Let l be an atom in s_t. We show that $l \cdot \theta$ is in $lgg(s_x, s_i)$ and that it is not removed by the restriction to σ. Let t be a term appearing in l. The matching σ contains the entry

$$\texttt{[}t \cdot \theta'_x \texttt{ - } t \cdot \theta'_i \texttt{ => } lgg(t \cdot \theta'_x, t \cdot \theta'_i)\texttt{]},$$

since t appears in s_t. The substitution θ contains $\{v \mapsto lgg(v \cdot \theta'_x, v \cdot \theta'_i)\}$ for every variable v appearing in $s_t \cup \{b_t\}$ (and thus for every variable in s_t), therefore $t \cdot \theta = lgg(t \cdot \theta'_x, t \cdot \theta'_i)$. Indeed, $lgg(t \cdot \theta'_x, t \cdot \theta'_i)$ appears in σ. The atom $l \cdot \theta$ appears in $lgg(s_t \cdot \theta'_x, s_t \cdot \theta'_i)$ and therefore in $lgg(s_x, s_i)$ since $s_t \cdot \theta'_x \subseteq s_x$, $s_t \cdot \theta'_i \subseteq s_i$ and $\theta = \{v \mapsto lgg(v \cdot \theta'_x, v \cdot \theta'_i) \mid v \text{ is a variable of } s_t\}$. Also, $l \cdot \theta$ appears in $lgg_{|\sigma}(s_x, s_i)$ since we have seen that any term in $l \cdot \theta$ appears in σ.

In our example the only $l \in s_t \cdot \theta$ is $p(g(c), x, f(y), z) \cdot \theta = p(g(c), X, f(Y), z)$. And $lgg_{|\sigma}(s_x, s_y)$ is precisely $\{p(g(c), X, f(Y), z)\}$.

– $ineq(s_t \cup \{b_t\}) \cdot \theta \subseteq ineq(s \cup c)$**:** We have to show that for any two distinct terms t, t' of $s_t \cup \{b_t\}$, the terms $t \cdot \theta$ and $t' \cdot \theta$ are also different terms in $s \cup c$, and therefore the inequality $t \cdot \theta \neq t' \cdot \theta$ appears in $ineq(s \cup c)$. By hypothesis, $ineq(s_t \cup \{b_t\}) \cdot \theta'_x \subseteq ineq(s_x \cup c_x)$. Since $\theta'_x = \theta \cdot \theta_x$, we get $ineq(s_t \cup \{b_t\}) \cdot \theta \cdot \theta_x \subseteq ineq(s_x \cup c_x)$ and so $t \cdot \theta \cdot \theta_x$ and $t' \cdot \theta \cdot \theta_x$ are different terms of $s_x \cup c_x$. From Property 5 in Lemma 1 it follows that $t \cdot \theta \neq t' \cdot \theta \in ineq(s \cup c)$.

– $b_t \cdot \theta \in c$**:** By hypothesis, $b_t \cdot \theta'_x \in c_x$. Also, $b_t \cdot \theta'_i \in Atoms_P(s_i \cup c_i)$ implies (because $[s_i, c_i]$ is full), that $b_t \cdot \theta'_i \in s_i \cup c_i$. Notice that $b_t \cdot \theta = lgg_{|\sigma}(b_t \cdot \theta'_x, b_t \cdot \theta'_i)$ by construction. Therefore $b_t \cdot \theta \in c = lgg_{|\sigma}(s_x, c_i) \cup lgg_{|\sigma}(c_x, s_i) \cup lgg_{|\sigma}(c_x, c_i)$. ■

Claim 44 $WSize([s, c]) < WSize([s_i, c_i])$.

Proof. By definition, we need to show that $WSize(s) < WSize(s_i)$ or ($WSize(s) = WSize(s_i)$ and $WSize(c) < WSize(c_i)$). By Lemma 34, we know that $|s| \leq |s_i|$, therefore $WSize(s) \leq WSize(s_i)$ — the *lgg* never substitutes a term by one of greater weight: either functional terms are substituted by variables or they remain the same. According to our definition of *WSize*, variables weigh less than functional terms.

If $WSize(s) < WSize(s_i)$, then the condition stated in this lemma is true. Otherwise, $WSize(s) = WSize(s_i)$. Lemma 34 shows that $|s \cup c| \leq |s_i \cup c_i|$. Since $|s| = |s_i|$, we conclude that $|c| \leq |c_i|$, and hence $WSize(c) \leq WSize(c_i)$ by the same argument as above. Thus, $s \cdot \theta_i = s_i$ and $s_i \cdot \theta_i^{-1} = s$. Again, we split the proof into two cases. If $WSize(c) < WSize(c_i)$ then the lemma is satisfied. Otherwise $WSize(c) = WSize(c_i)$, and $[s, c]$, $[s_i, c_i]$ are syntactic variants. The following reasoning arrives to a contradiction, disproving this case. Since $[s, c]$ and $[s_i, c_i]$ are variable renamings, $c \cdot \theta_i = c_i$ and $c_i \cdot \theta_i^{-1} = c$. By the previous claim, it holds that $b_t \cdot \theta \in c$ and therefore there exists a b_i s.t. $b_i = b_t \cdot \theta \cdot \theta_i \in c_i$. The substitutions θ_i and θ'_x are variable renamings, and (by previous claim) $\theta'_x = \theta \cdot \theta_x$, therefore the substitution $\hat{\theta} = \theta_i^{-1} \cdot \theta_x$ is well defined and is a variable renaming. It follows that

$s_i \cdot \hat{\theta} \subseteq s_x$ and $b_i \cdot \hat{\theta} = \underbrace{b_t \cdot \theta \cdot \theta_i}_{b_i} \cdot \underbrace{\theta_i^{-1} \cdot \theta_x}_{\hat{\theta}} = b_t \cdot \theta \cdot \theta_x = b_t \cdot \theta'_x \in c_x$ (by assumption). Therefore, $H \models s_i \rightarrow b_i \models s_i \cdot \hat{\theta} \rightarrow b_i \cdot \hat{\theta} \models s_x \rightarrow b_x$ (where $b_x = b_t \cdot \theta'_x \in c_x$) contradicting the fact that $[s_x, c_x]$ is a counterexample. ■

This finally completes the proof of Lemma 38. ■

Corollary 45 *If a counterexample $[s_x, c_x]$ is appended to S, it is because there is no element in S capturing a clause in $U(T)$ that is also captured by $[s_x, c_x]$.* ■

Lemma 46 *Every time the algorithm is about to make an equivalence query, it is the case that every meta-clause in S captures at least one of the clauses of $U(T)$ and every clause of $U(T)$ is captured by at most one meta-clause in S.*

Proof. All meta-clauses included in S are full by Corollary 37. By construction, their consequents are non-empty so that we can apply Lemma 22, and conclude that all counterexamples in S capture some clause of $U(T)$.

An induction on the number of iterations of the main loop in line 2 of Learn-Horn-Closed shows that no two different meta-clauses in S capture the same clause of $U(T)$. In the first loop the lemma holds trivially (there are no elements in S). By the induction hypothesis we assume that the lemma holds before a new iteration of the loop. We see that after completion of that iteration of the loop the lemma must also hold. Two cases arise.

The minimized counterexample $[s_x, c_x]$ is appended to S. By Corollary 45, we know that $[s_x, c_x]$ does not capture any clause in $U(T)$ also captured by some element $[s_i, c_i]$ in S. This, together with the induction hypothesis, assures that the lemma is satisfied in this case.

Some $[s_i, c_i]$ is replaced in S. We denote the updated sequence by S' and the updated element in S' by $[s'_i, c'_i]$. The induction hypothesis claims that the lemma holds for S. We have to prove that it also holds for S' as updated by the algorithm. Assume it does not. The only possibility is that the new element $[s'_i, c'_i]$ captures

some clause of $U(T)$, say $s_t \stackrel{\neq}{\rightarrow} b_t$ also captured by some other element $[s_j, c_j]$ of S', with $j \neq i$. The meta-clause $[s'_i, c'_i]$ is a basic pairing of $[s_x, c_x]$ and $[s_i, c_i]$, and hence it is also legal. Applying Lemma 35 we conclude that one of $[s_x, c_x]$ or $[s_i, c_i]$ captures $s_t \stackrel{\neq}{\rightarrow} b_t$.

Suppose $[s_i, c_i]$ captures $s_t \stackrel{\neq}{\rightarrow} b_t$. This contradicts the induction hypothesis, since both $[s_i, c_i]$ and $[s_j, c_j]$ appear in S and capture $s_t \stackrel{\neq}{\rightarrow} b_t$ in $U(T)$.

Suppose $[s_x, c_x]$ captures $s_t \stackrel{\neq}{\rightarrow} b_t$. If $j < i$, then $[s_x, c_x]$ would have refined $[s_j, c_j]$ instead of $[s_i, c_i]$ (Lemma 38). Therefore, $j > i$. But then we are in a situation where $[s_j, c_j]$ captures a clause also covered by $[s_i, c_i]$. By Corollary 36, all meta-clauses in position i cover $s_t \stackrel{\neq}{\rightarrow} b_t$ during the history of S. Consider the iteration in which $[s_j, c_j]$ first captured $s_t \stackrel{\neq}{\rightarrow} b_t$. This could have happened by appending the counterexample $[s_j, c_j]$, which contradicts Lemma 38 since $[s_i, c_i]$ or an ancestor of it was covering $s_t \stackrel{\neq}{\rightarrow} b_t$ but was not replaced. Or it could have happened by refining $[s_j, c_j]$ with a pairing of a counterexample capturing $s_t \stackrel{\neq}{\rightarrow} b_t$. But then, by Lemma 38 again, the element in position i should have been refined, instead of refining $[s_j, c_j]$. ■

5.2.9 Deriving the complexity bounds

Recall that c' stands for the number of clauses in the transformation $U(T)$ and that by Lemma 18, $c' \leq ct^v$, where t and v are upper bounds on the number of terms and variables in each clause in T and c is the number of clauses in T. By Lemma 46 the number of clauses in $U(T)$ bounds the number of elements in S, and therefore:

Corollary 47 $|S| \leq c'$. ■

What follows is a detailed account of the number of queries made in every procedure. Remember that we use the following parameters controlling the complexity of the target expression T:

- p: number of predicate symbols in the signature.

- a: upper bound on the arity of predicate and function symbols.
- v: upper bound on the number of distinct variables per clause.
- t: upper bound on the number of distinct terms per clause.
- c: number of clauses.

The complexity of the algorithm also depends on the complexity of the counterexamples received. To account for this, we use the parameter $\hat{t}$ to denote an upper bound on the number of distinct terms present in a clause returned by the equivalence query oracle as counterexample.

Lemma 48 *If $[s_x, c_x]$ is a minimized counterexample, then, $|s_x \cup c_x| \leq pt^a$.*

Proof. By Corollary 30, there are a maximum of t terms in a minimized counterexample. There are a maximum of pt^a different atoms built up from t terms if p is the number of predicate symbols in the signature and a is an upper bound on their arity. ■

Lemma 49 *The algorithm makes $O(c'pt^a)$ equivalence queries.*

Proof. Notice that any set of atoms containing t distinct terms can be generalized at most t times. This is because after generalizing a term into a variable, it cannot be further generalized. The sequence S has at most c' elements. The following actions can happen after refining a meta-clause in S (possibly combined): either (1) one atom is dropped from the antecedent, or (2) an atom moves from antecedent to consequent, or (3) an atom is dropped from the consequent, or (4) some term is generalized. This can happen $c'pt^a$ times for (1), $c'pt^a$ times for (2), $c'pt^a$ times for (3), and $c't$ times for (4), that is $c'(t + 3pt^a)$ in total. We need c' extra calls to add all the counterexamples to S. In total $\underline{c'}(1 + t + 3\underline{pt^a}) = O(c'pt^a)$. ■

Lemma 50 *The algorithm makes $O(p\hat{t}^{a+1})$ membership queries during the minimization procedure.*

Proof. Let $B \to b$ be the clause input to MINIMIZE. To compute the first version of the full meta-clause we need to test the $p\hat{t}^a$ possible atoms built up from $\hat{t}$ distinct terms appearing in $B \to b$. Therefore, we make $p\hat{t}^a$ initial calls. Notice that MINIMIZE never introduces new terms, and hence $\hat{t}$ remains an upper bound on the number of terms of the clause under construction for the duration of the process. Next, we note that the first version of c_x has at most $p\hat{t}^a$ atoms. The first loop (generalization of terms) is executed at most $\hat{t}$ times, one for every term appearing in the first version of $[s_x, c_x]$. In every execution, at most $|c_x| \leq p\hat{t}^a$ membership calls are made. In this loop there are a total of $p\hat{t}^{a+1}$ calls. The second loop of the minimization procedure is also executed at most $\hat{t}$ times, one for every term in $[s_x, c_x]$. Again, since at most $p\hat{t}^a$ calls are made in the body on this second loop, the total number of calls is bounded by $p\hat{t}^{a+1}$. This makes a total of $p\hat{t}^a + 2p\hat{t}^{a+1} = O(p\hat{t}^{a+1})$. ■

Lemma 51 *The algorithm makes at most pt^a membership queries to check the validity of a basic pairing in line 4 of* LEARN-CLOSED-HORN.

Proof. Let $[s, c] \in$ BASIC-PAIRINGS$([s_x, c_x], [s_i, c_i])$ be any basic pairing to be validated as a successful replacement in line 4 of LEARN-CLOSED-HORN. By Lemmas 34 and 48, we conclude that $|c| \leq |s_x \cup c_x| \leq pt^a$. ■

Lemma 52 *The algorithm makes $O(c's^2t^a\hat{t}^{a+1} + c'^2s^2t^{2a+k})$ membership queries.*

Proof. The main loop is executed as many times as equivalence queries are made. In every loop, the minimization procedure is executed once and for every element in S, a maximum of t^v basic pairings are checked.

This is:

$$\underbrace{pc't^a}_{\#iterations} \times\{\underbrace{p\hat{t}^{a+1}}_{minim.} + \underbrace{c'}_{|S|} \cdot \underbrace{t^v}_{\#pairings} \cdot \underbrace{pt^a}_{pairing}\} = O(c'p^2t^a\hat{t}^{a+1} + c'^2p^2t^{2a+v}).$$

■

We arrive at our main theorem:

Theorem 53 LEARN-CLOSED-HORN *exactly identifies every closed Horn expression making* $O(c'pt^a)$ *equivalence queries and* $O(c'p^2t^a\hat{t}^{a+1} + c'^2p^2t^{2a+v})$ *membership queries. Furthermore, the running time is polynomial in* $c' + p + t^v + t^a + \hat{t}^a$. ■

Since $c' \leq ct^v$, we obtain:

Corollary 54 LEARN-CLOSED-HORN *exactly identifies every closed Horn expression making* $O(cpt^{a+v})$ *equivalence queries and* $O(cp^2t^{a+v}\hat{t}^{a+1} + c^2p^2t^{2a+3v})$ *membership queries. Furthermore, the running time is polynomial in* $c + p + t^v + t^a + \hat{t}^a$. ■

Corollary 55 *Assume the parameters* p, a *identifying the signature are constant. Assume that the number of distinct terms in any given counterexample is upper bounded by* t. *Then,* LEARN-CLOSED-HORN *exactly identifies every closed Horn expression making* $O(ct^{a+v})$ *equivalence queries and* $O(c^2t^{2a+3v})$ *membership queries. Furthermore, the running time is polynomial in* $c + t^v + t^a$. ■

5.3 Fully inequated closed Horn expressions

In this section we study the learnability of the class of fully inequated closed Horn expressions. Clauses in expressions in this class are fully inequated, that is, the antecedent of every clause contains all possible combinations of the atom $t \neq t'$, for each term t and t' appearing in the clause. As a consequence, the antecedent of a fully inequated clause is only satisfied by a given interpretation I, if every term in the clause is mapped to a different object of I's domain. As an example, take the clause $human(father(x)) \wedge human(mother(x)) \rightarrow human(x)$. Its intended meaning is clearly that $x \neq father(x) \neq mother(x)$, and hence this clause can be assumed to be fully inequated. As expected, we say that a meta-clause is fully inequated if its antecedent contains all possible inequalities between terms appearing in the meta-clause.

For any given expression T, the transformation $U(T)$ described in Section 5.2.1 is fully inequated by construction. We used $U(\cdot)$ as a trick in the proof of correctness to help quantify how long it takes the algorithm LEARN-CLOSED-HORN to terminate. If the target expression T is fully inequated itself, then $U(T) = T$ and this trick is not necessary. Moreover, the complexity bounds derived are better since the blow-up in the number of clauses from T to $U(T)$ does not occur.

In the remaining of this chapter, we describe the (slight) changes that we need to make to LEARN-CLOSED-HORN in order to learn the more restricted class of fully inequated closed Horn expressions with better bounds. The proof of correctness is omitted as it is very similar to the one presented for LEARN-CLOSED-HORN. Complete details and proof for the case of learning range restricted Horn expressions can be found in (Arias and Khardon, 2000). The only modifications needed are in the procedures MINIMIZE and PAIRING.

Minimization. The purpose of the minimization procedure is to produce a meta-clause containing as few terms as possible while maintaining the property that it is still a counterexample. In addition, now we want the counterexample to be fully inequated. The changes proposed here are to guarantee that the counterexample $[s_x, c_x]$ as it is being minimized is fully inequated during all stages of the process. Assume that the counterexample $A \rightarrow a$ given by the equivalence query oracle is fully inequated. The first version of the minimized meta-clause $[s_x, c_x]$ after line 1 of MINIMIZE is fully inequated — simply because $A \rightarrow a$ is — so no change there is needed. However, this is not true for subsequent updates of $[s_x, c_x]$. The effect of generalizing and dropping terms (lines 2 and 3 of MINIMIZE) is that of removing atoms. The result of removing atoms from a fully inequated meta-clause need not be fully inequated. More precisely, we might end up with terms in inequalities that do not appear anywhere else in the resulting meta-clause. As an example, suppose $[s_x, c_x] = [\{(a \neq b), (b \neq c), (a \neq c), p(a, b), q(c)\}, \{r(b), r(c)\}]$. If we drop the constant b, then the resulting $[s'_x, c'_x] = [\{(a \neq c), q(c)\}, \{r(c)\}]$. Notice that

term a is still among s'_x inequalities but it does not appear anywhere else. To avoid this, we artificially need to fix the inequalities of the resulting $[s'_x, c'_x]$ to guarantee that it is fully inequated, done by the following procedure:

FULLY-INEQUATED-FIX$([s, c])$

1 $s' \leftarrow s \cap Atoms_P(s \cup c)$

2 **return** $[ineq(s' \cup c) \cup s', c]$

The resulting minimization procedure is:

FULLY-INEQUATED-MINIMIZE$(H, A \rightarrow a)$

1 $[s_x, c_x] \leftarrow$ CONS-CLOSURE(ANT-CLOSURE$(H, [A, \{a\}])$)

2 **for** every functional term t in $s_x \cup c_x$, in decreasing order of size

do Let $[s'_x, c'_x]$ be the meta-clause obtained from $[s_x, c_x]$ after substituting all occurrences of the term t by a new variable x_t

$[s'_x, c'_x] \leftarrow$ FULLY-INEQUATED-FIX$([s'_x, c'_x])$

if RHS$(s'_x, c'_x) \neq \emptyset$

then $[s_x, c_x] \leftarrow [s'_x,$ RHS$(s'_x, c'_x)]$

3 **for** every term t in $s_x \cup c_x$, in increasing order of size

do Let $[s'_x, c'_x]$ be the meta-clause obtained after removing from $[s_x, c_x]$ all those atoms containing t

$[s'_x, c'_x] \leftarrow$ FULLY-INEQUATED-FIX$([s'_x, c'_x])$

if RHS$(s'_x, c'_x) \neq \emptyset$

then $[s_x, c_x] \leftarrow [s'_x,$ RHS$(s'_x, c'_x)]$

4 **return** $[s_x, c_x]$

Note that the only difference w.r.t. MINIMIZE is the one line that fixes the meta-clause $[s'_x, c'_x]$ both after generalizing and dropping terms.

Pairing. Given a matching σ and two meta-clauses $[s_x, c_x]$ and $[s_i, c_i]$, its pairing $[s, c]$ is computed in the new algorithm as:

FULLY-INEQUATED-PAIRING$(\sigma, [s_x, c_x], [s_i, c_i])$

1 $s'_x \leftarrow s_x \cap Atoms_P(s_x)$

2 $s'_i \leftarrow s_i \cap Atoms_P(s_i)$

3 $[s, c] \leftarrow$ PAIRING$(\sigma, [s'_x, c_x], [s'_i, c_i])$

4 **return** FULLY-INEQUATED-FIX$([s, c])$

That is, we ignore the inequalities, compute the "normal" pairing, and then we add all the inequalities needed at the end. Finally our learnability result is:

Theorem 56 *The modified algorithm learns the class of fully inequated closed Horn expressions making $O(ct^a)$ calls to the equivalence oracle and $O(c^2t^{2a+v})$ to the membership oracle. Furthermore, the running time is polynomial in $c + t^v$.*

It is important to observe the reduction in the number of queries made by the modified algorithm. This is particularly significant for the number of equivalence queries. This number is reduced to a polynomial if a is considered constant. The number of membership queries is also reduced, although the exponential dependencies remain unchanged.

Chapter 6

The VC Dimension

The remainder of this thesis is concerned with finding lower bounds on the problem of exactly learning first order closed Horn expressions from membership and equivalence queries. In fact, the lower bounds developed here hold even for first order Horn expressions which are both range restricted and constrained.

This chapter characterizes the Vapnik-Chervonenkis dimension (*VCDim*) of first order Horn expressions. It is known that the VC Dimension provides tight bounds on the number of examples for PAC learning (Ehrenfeucht et al., 1989) as well as a lower bound for the number of equivalence and membership queries for exact learning (Maass and Turán, 1992). We parameterize the class of first order Horn expressions with the parameters c, t, and l that stand respectively for number of clauses, maximum number of (distinct) terms per clause, and maximum number of literals per clause.

It is well known that for a finite class $\mathcal{T}$, we have $VCDim(\mathcal{T}) \leq \log|\mathcal{T}|$. In Section 4.2.3 we show that $DAGSize(E) = O(ct + cl)$ for every first order Horn expression E. Hence, if $\mathcal{H}^{\leq c,t,l}$ is the class of first order Horn expressions with at most c clauses, at most t terms per clause, and at most l literals per clause, then $\left|\mathcal{H}^{\leq c,t,l}\right| \leq 2^{\tilde{O}(ct+cl)}$, where $\tilde{O}(\cdot)$ is used to hide logarithmic factors. From this we can conclude that $VCDim(\mathcal{H}^{\leq c,t,l}) = \tilde{O}(ct+cl)$. The rest of this chapter shows that $VCDim(\mathcal{H}^{\leq c,t,l}) = \Omega(cl + ct)$.

We start with the necessary definitions (Blumer et al., 1989).

Definition 29 Let $\mathcal{I}$ be a set, $\mathcal{H} \subseteq 2^{\mathcal{I}}$, and $\mathcal{S} \subseteq \mathcal{I}$. Then $\Pi_{\mathcal{H}}(\mathcal{S}) = \{h \cap \mathcal{S} \mid h \in \mathcal{H}\}$ is the set of subsets of $\mathcal{S}$ that can be obtained by intersection with elements of $\mathcal{H}$. If $|\Pi_{\mathcal{H}}(\mathcal{S})| = 2^{|\mathcal{S}|}$, then we say that $\mathcal{H}$ shatters $\mathcal{S}$. Finally, $VCDim(\mathcal{H})$ is the size of the largest set shattered by $\mathcal{H}$ (or ∞ if arbitrary large sets are shattered).

In our case $\mathcal{I}$ is a set of interpretations, and $\mathcal{H}$ is some class of first order Horn expressions interpreted under $\models$. We identify every $h \in \mathcal{H}$ with the set of interpretations that satisfy h. Hence, the lower bounds that follow from our constructions apply to the setting of learning from interpretations only.

In Theorems 57 through 61 we construct sets of interpretations of appropriate cardinality, and show how to shatter them by giving families of first order Horn expressions separating each possible dichotomy of the interpretation sets. In our constructions we make extensive use of the function mappings in the interpretations to ensure that terms evaluate to appropriate values in the interpretations so that separation is guaranteed.

Theorem 57 *There exists a set of c interpretations that can be shattered using first order Horn expressions bounded by $NClauses \leq c$, $NTerms \leq \log c + 3$, $NLits = 2$, $NVars = 0$, $Depth = \log c$, $Arity = 2$, $NFuncs = 4$ and $NPreds = 2$.*

Proof. We construct a set of c different terms using a function f of arity 2 and three constants 1, 2 and 3 and by forming ground terms of depth $\log c$ in the following manner:

$$\hat{\mathcal{T}} = \{f(a_1, f(a_2, f(a_3, f(...f(a_{\log c}, 3)...)) \mid a_i \in \{1, 2\} \text{ for all } 1 \leq i \leq \log c\}$$

Notice that there are exactly $2^{\log c} = c$ such terms. Moreover, every term in $\hat{\mathcal{T}}$ is of size $2 \log c + 1$ and contains at most $\log c + 3$ distinct subterms.

Each interpretation $I_{\hat{t}}$ in the set of interpretations $\mathcal{I}$ to be shattered contains in its extension a single atom $P(\hat{t})$ where $\hat{t} \in \hat{\mathcal{T}}$. Hence, $|\mathcal{I}| = |\hat{\mathcal{T}}| = c$. In addition, the

domain of the interpretation $I_{\hat{t}}$, consists of the $\Theta(\log c)$ objects corresponding to the subterms appearing in $\hat{t}$ (including itself) and a distinguished object $*$. The function mapping for f is defined to follow the functional structure of the distinguished term $\hat{t}$, undefined entries are mapped to $*$. Notice that any term $t' \in \hat{\mathcal{T}}$ s.t. $\hat{t} \neq t'$ is mapped to the special object $*$ under the interpretation $I_{\hat{t}}$.

Now, we define the Horn expression $H_{\mathcal{S}}$ using predicate symbols $P/1$ and $F/0$ that separates any arbitrary subset $\mathcal{S} \subseteq \mathcal{I}$ as

$$H_{\mathcal{S}} = \left\{ P(\hat{t}) \rightarrow F() \;\middle|\; I_{\hat{t}} \in \mathcal{S} \right\}.$$

Any interpretation in $\mathcal{S}$ falsifies one of the clauses in $H_{\mathcal{S}}$, and hence falsifies the whole Horn expression; any interpretation not in $\mathcal{S}$ falsifies every clause's antecedent in $H_{\mathcal{S}}$ since the term present in the clause is mapped to the special object $*$ which does not appear in any of the interpretations' extension. ■

A VC Dimension construction of (Khardon, 1999a) uses a signature that grows with $NTerms$. The following theorem modifies this construction to use a fixed signature.

Theorem 58 *For $l \leq t^a$, there exists a set of l interpretations that can be shattered using first order Horn expressions bounded by $NTerms = 2t$, $Nvars \leq t$, $Depth = \log t$, $NLits \leq l$, $NPreds = 3$, $NFuncs = 1$, $Arity \leq a$ and $NClauses = 1$.*

Proof. We construct a set of interpretations $\mathcal{I}$ that is shattered using first order Horn expressions with parameters as stated. Fix a and t. The expressions use a 0-ary predicate $F()$, a unary predicate L and a predicate symbol Q of arity $\log_t l$. Let

$$Q_{all} = \left\{ Q(i_1, \ldots, i_{\log_t l}) \;\middle|\; i_j \in \{1, .., t\} \text{ for all } j = 1, \ldots, \log_t l \right\}.$$

Notice that $|Q_{all}| = t^{\log_t l} = l$.

Let f be a binary function, and let $\hat{t}$ be the term represented by a binary balanced tree of depth $\log t$ whose leaves are labeled by the objects $1 \ldots t$ (in order)

and whose internal nodes are labeled by the function symbol f. Such a term contains $2t$ subterms. The domain for all the interpretations in $\mathcal{I}$ includes objects $\{1, .., t\}$, an object for each subterm of $\hat{t}$, and a special object $*$. The function mappings for f follow the functional structure of $\hat{t}$ with undefined entries completed by the special domain object $*$. Interpretations include in their extension the atom $L(\hat{t})$ and all the atoms in Q_{all} except one. Hence, there are l interpretations in $\mathcal{I}$.

The expression that separates an arbitrary $\mathcal{S} \subseteq \mathcal{I}$ is $\mathcal{H}_{\mathcal{S}} = C_{\mathcal{S}} \rightarrow F()$, where $F()$ is a nullary predicate symbol and $C_{\mathcal{S}}$ is the intersection of the $Q()$ atoms in the extensions of all the interpretations in $\mathcal{S}$ plus the atom $L(\hat{t})$ after substituting every domain object $j \in \{1, .., t\}$ by a corresponding variable x_j.

Suppose $I \in \mathcal{S}$. Take the substitution $\{x_j \mapsto j\}$. Then I falsifies $\mathcal{H}_{\mathcal{S}}$ because its antecedent $C_{\mathcal{S}}$ is satisfied (it is a subset of the extension of I) and its consequent $F()$ is falsified. Suppose on the other hand that $I \notin \mathcal{S}$. Substitutions other than $\{x_j \mapsto j\}$ falsify the antecedent of $\mathcal{H}_{\mathcal{S}}$ because of the atom $L(\hat{t})$. The clause $\mathcal{H}_{\mathcal{S}}$ is satisfied under the substitution $x_j \mapsto j$ because the "omitted Q" in I's extension is present in $C_{\mathcal{S}}$. ■

Theorem 59 *For $l \leq t^a$, there exists a set of cl interpretations that can be shattered using first order Horn expressions bounded by $NClauses \leq c$, $NTerms = \Theta(\log c + t)$, $NLits \leq l$, $NVars \leq t$, $Depth = \Theta(\log c + \log t)$, $Arity \leq a$, $NFuncs = 5$ and $NPreds = 3$.*

Proof. Let $\mathcal{I}$ be the set shattered in Theorem 58. We create a new set of interpretations $\mathcal{I}^+$ of cardinality cl in the following way. We have an additional set of c terms constructed in the same way as in Theorem 57, let us denote this set $\hat{\mathcal{T}}_c$. As in Theorem 57, $\hat{\mathcal{T}}_c$ contains c distinct terms of depth $\log c$ each.

We augment the interpretations in the construction of Theorem 58 by associating $I \in \mathcal{I}$ with a new term in $\hat{\mathcal{T}}_c$ (and hence we create c new interpretations in $\mathcal{I}^+$ for each old interpretation in $\mathcal{I}$), adding $\log c$ new objects and the corresponding functional mappings following the terms' structure, completing undefined entries

with the special object $*$. Additionally, we include the atom $F(*)$ in each of the interpretations' extensions (notice that a term c' evaluates to $*$ in the interpretations which do not have c' as their distinguished term). Hence $|\mathcal{I}^+| = cl$.

The new expression separating an arbitrary subset $\mathcal{S} \subseteq \mathcal{I}^+$ is $H_{\mathcal{S}}$:

$$\left\{ C_{\mathcal{S}_{\hat{c}}} \rightarrow F(\hat{c}) \;\middle|\; \hat{c} \in \hat{\mathcal{T}}_c \right\},$$

where $\mathcal{S}_{\hat{c}}$ is the subset of interpretations in $\mathcal{S}$ with distinguished term $\hat{c}$ and $C_{\mathcal{S}_{\hat{c}}}$ is constructed as in Theorem 58.

We finally prove that I falsifies $H_{\mathcal{S}}$ iff $I \in \mathcal{S}$. Suppose that $\hat{c}$ is the distinguished term in $\hat{\mathcal{T}}_c$ associated to I. Terms $c' \neq \hat{c}$ evaluate to $*$ under I, and every clause with consequent other than $F(\hat{c})$ in $H_{\mathcal{S}}$ is hence satisfied. The clause containing $F(\hat{c})$ is falsified iff $I \in \mathcal{S}_{\hat{c}}$ by the same reasoning as in Theorem 58. ■

The next result shows that by varying the number of terms we can shatter arbitrarily large sets with a fixed signature.

Theorem 60 *There exists a set of t interpretations that can be shattered using Horn expressions bounded by $NClauses = 1$, $NTerms \leq 4t$, $NLits = 2$, $NVars = 0$, $Depth = 2\log t + 2$, $Arity = 2$, $NFuncs \leq 9$ and $NPreds = 2$.*

Proof. Let $t = k \log k$ for some $k \in N$. Using the same signature as in Theorem 57 we generate a set $\hat{\mathcal{T}}$ of k terms of depth $\log k$ each. We associate to every interpretation a term in $\hat{\mathcal{T}}$ and an index $i \in \{1, .., \log k\}$ and we denote by $I_{\hat{t},i}$ the interpretation associated to $(\hat{t}, i) \in \hat{\mathcal{T}} \times \{1, .., \log k\}$. Thus, we have a set of interpretations $\mathcal{I}$ s.t. $|\mathcal{I}| = |\hat{\mathcal{T}}|\,|\{1, .., \log k\}| = k \log k = t$.

Given a subset $\mathcal{S} \subseteq \mathcal{I}$, we construct a big term $TREE_{\mathcal{S}}$ which intuitively associates to every possible term $\hat{t}$ in $\hat{\mathcal{T}}$ a set of indices $l_{\hat{t}}$ where $l_{\hat{t}} = \left\{ i \;\middle|\; I_{\hat{t},i} \in \mathcal{S} \right\}$. We then appropriately define the function mappings in each interpretation $I_{\hat{t},i}$ so that the term $TREE_{\mathcal{S}}$ evaluates to a special domain object y iff index i appears in the set of indices for term $\hat{t}$ encoded in $TREE_S$. Each interpretation includes in its

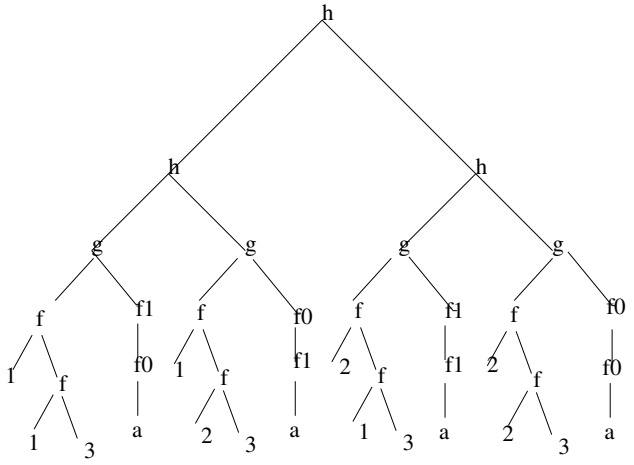
extension the atom $M(y)$ so that the clause

$$H_{\mathcal{S}} = M(TREE_{\mathcal{S}}) \rightarrow F()$$

is falsified by interpretation I iff the term $TREE_{\mathcal{S}}$ evaluates to y under I.

We first describe the structure of the term $TREE_{\mathcal{S}}$. Let $\mathcal{S}_{\hat{t}}$ be the subset of $\mathcal{S}$ consisting of interpretations $I_{\hat{t},i}$ in $\mathcal{S}$ and let $l_{\hat{t}} = \left\{ i \,\middle|\, I_{\hat{t},i} \in \mathcal{S}_{\hat{t}} \right\}$. We encode the set $l_{\hat{t}}$ with the term $f_{i_1}(f_{i_2}(\cdots f_{i_{\log k}}(a))\cdots)$ where $i_j = 0$ if $j \notin l_{\hat{t}}$ and $i_j = 1$ otherwise. Denote this term by $t_{l_{\hat{t}}}$. As an example, assume $\log k = 6$ and let the set $l_{\hat{t}} = \{1, 4, 5\}$. Then, $t_{l_{\hat{t}}} = f_1(f_0(f_0(f_1(f_1(f_0(a))))))$. Notice that we are using two unary functions f_0 and f_1 and a constant a. Next we use a binary function g to encode the association between terms $\hat{t}$ and their sets of indices $l_{\hat{t}}$ as $g(\hat{t}, t_{l_{\hat{t}}})$. Finally, $TREE_{\mathcal{S}}$ is constructed as a balanced tree (using binary function h) whose leaves are terms of the form $g(\hat{t}, t_{l_{\hat{t}}})$, for every $\hat{t} \in \hat{\mathcal{T}}$. As an example, suppose $k = 4$. Then $\hat{\mathcal{T}} = \{\hat{t}_1, \hat{t}_2, \hat{t}_3, \hat{t}_4\}$, where $\hat{t}_1 = f(1, f(1, 3))$, $\hat{t}_2 = f(1, f(2, 3))$, $\hat{t}_3 = f(2, f(1, 3))$ and $\hat{t}_4 = f(2, f(2, 3))$. Suppose $\mathcal{S} = \{(\hat{t}_1, 1), (\hat{t}_2, 2), (\hat{t}_3, 1), (\hat{t}_3, 2)\}$. Then:

- $l_{\hat{t}_1} = \{1\}$, $l_{\hat{t}_2} = \{2\}$, $l_{\hat{t}_3} = \{1, 2\}$ and $l_{\hat{t}_4} = \{\}$.
- $t_{l_{\hat{t}_1}} = f_1(f_0(a))$, $t_{l_{\hat{t}_2}} = f_0(f_1(a))$, $t_{l_{\hat{t}_3}} = f_1(f_1(a))$ and $t_{l_{\hat{t}_4}} = f_0(f_0(a))$.
- $TREE_{\mathcal{S}} =$



Let us now describe in detail the domain and function mappings for interpretation $I_{\hat{t},i}$. The domain objects are:

- Three special objects $*, y, n$.
- Up to $\log k + 3$ distinct objects that represent all terms and subterms present in the distinguished term $\hat{t}$.
- Up to $2k + 1$ objects representing all the possible terms and subterms of the vector indices $f_{i_1}(f_{i_2}(\cdots f_{i_{\log k}}(a))\cdots)$ for all possible $i_j \in \{0, 1\}$ where $1 \leq j \leq \log k$.

The function mappings are defined as follows:

- The constants $1, 2, 3$ potentially appearing in $\hat{t}$ are mapped to objects $1, 2, 3$. The mapping for binary function f follows functional structure of $\hat{t}$, with undefined entries mapped to the special object $*$.
- The constant a is mapped to object a. Unary functions f_0 and f_1 also mimic the functional structure of terms and subterms of $f_{i_1}(f_{i_2}(\cdots f_{i_{\log k}}(a))\cdots)$ for all possible $i_j \in \{0, 1\}$ where $1 \leq j \leq \log k$.
- The binary function $g(t_1, t_2)$ is mapped to special object y iff $t_1 = \hat{t}$ and the unary function used at depth i in term t_2 is f_1. Otherwise it is set to the special object n.
- Finally, the binary function $h(a1, a2)$ is mapped to domain object y iff either $a1 = y$ or $a2 = y$, otherwise it is mapped to object n.

Finally, the only atom true in each interpretation is $M(y)$.

We prove that $I_{\hat{t},i}$ falsifies $H_{\mathcal{S}}$ iff $I_{\hat{t},i} \in \mathcal{S}$. Notice that $I_{\hat{t},i}$ falsifies $H_{\mathcal{S}}$ iff $I_{\hat{t},i}$ satisfies the atom $M(TREE_{\mathcal{S}})$ iff the term $TREE_{\mathcal{S}}$ is mapped to the domain object y under $I_{\hat{t},i}$ iff some term $g(t_1, t_2)$ is mapped to y iff term $g(\hat{t}, t_2)$ is mapped to y (other terms $g(t_1, t_2)$ where $t_1 \neq \hat{t}$ are mapped to n by construction) iff the unary function used at depth i in term t_2 is f_1 iff $I_{\hat{t},i} \in \mathcal{S}$.

We finally quantify the complexity of the parameters used in $H_{\mathcal{S}}$: it has 1 clause, 2 literals, no variables, uses one single term of depth $\Theta(\log k)$ (that is $O(\log t)$) which contains $\Theta(k \log k)$ subterms (that is $\Theta(t)$ subterms) that are built from 4 constants, 5 function symbols whose maximal arity is 2. ■

Theorem 61 *There exists a set of ct interpretations that can be shattered using Horn expressions bounded by* $NClauses \leq c$, $NTerms = \Theta(t + \log c)$, $NLits = 2$, $NVars = 0$, $Depth = O(\log t + \log c)$, $Arity = 2$, $NFuncs \leq 9$ *and* $NPreds = 3$.

Proof. We extend the previous construction. Let $\mathcal{I}$ be the set shattered in Theorem 60. We create a new set of interpretations $\mathcal{I}^+$ of cardinality ct in the following way. We have an additional set of c terms constructed in the same way as in Theorem 57 but using as constants 1,2,3 and as binary function g; let us denote this set $\hat{\mathcal{T}}_c$. As in Theorem 57, $\hat{\mathcal{T}}_c$ contains c distinct terms of depth $\log c$ each. Notice that we can safely re-use the constants 1,2,3 and the function g since these are not combined in the previous construction.

As before, we augment the interpretations in the construction of Theorem 60 by associating $I \in \mathcal{I}$ with a new term in $\hat{\mathcal{T}}_c$ (and hence we create c new interpretations in $\mathcal{I}^+$ for each old interpretation in $\mathcal{I}$), adding $\log c$ new objects and the corresponding functional mappings following the term's structure. Hence $|\mathcal{I}^+| = ct$. In addition we modify the predicate M which now has arity 2. The only atom true in I is $M(\hat{c}, y)$ where $\hat{c}$ is the distinguished term in $\hat{\mathcal{T}}_c$ associated with I.

The new expression separating an arbitrary subset $\mathcal{S} \subseteq \mathcal{I}^+$ is:

$$H_{\mathcal{S}} = \left\{ M(\hat{c}, TREE_{\mathcal{S}_{\hat{c}}}) \rightarrow F() \;\middle|\; \hat{c} \in \hat{\mathcal{T}}_c \right\},$$

where $\mathcal{S}_{\hat{c}}$ is the subset of interpretations in $\mathcal{S}$ with distinguished term $\hat{c}$.

We finally prove that I falsifies $H_{\mathcal{S}}$ iff $I \in \mathcal{S}$. Suppose that $\hat{c}$ is the distinguished term in $\hat{\mathcal{T}}_c$ associated to I. I contains the atom $M(\hat{c}, y)$ in its extension, and every clause $M(c', TREE_{\mathcal{S}_{c'}}) \rightarrow F()$ in $H_{\mathcal{S}}$ s.t. $\hat{c} \neq c'$ is satisfied since term c' does not

evaluate to domain object $\hat{c}$ under I. The clause $M(\hat{c}, TREE_{\mathcal{S}_{\hat{c}}}) \rightarrow F()$ is falsified iff $I \in \mathcal{S}_{\hat{c}}$ by the same reasoning as in Theorem 60. ■

It is not hard to see that the constructions given above can be modified by adding dummy arguments in the antecedent and consequent so that the expressions used to shatter the given sets are both range restricted and constrained. For example, in the construction of the proof of Theorem 61, we can use as separating first order Horn expression (now range restricted and constrained):

$$H_{\mathcal{S}} = \left\{ M(\hat{c}, TREE_{\mathcal{S}_{\hat{c}}}) \rightarrow F(\hat{c}, TREE_{\mathcal{S}_{\hat{c}}}) \;\middle|\; \hat{c} \in \hat{\mathcal{T}}_c \right\},$$

and regardless to what the terms $\hat{c}$ and $TREE_{\mathcal{S}_{\hat{c}}}$ evaluate, they are always false since no atom with predicate symbol F is present in any of the interpretations' extensions. Similar observations hold for the other constructions. Thus we get:

Corollary 62 *Let $\mathcal{S}$ be a signature with at least 9 function symbols, 3 predicates and arity at least 2. The VC Dimension of the class of range restricted and constrained first order Horn expressions over $\mathcal{S}$ with at most c clauses, each using up to l literals and $t + \log c$ terms is $\Omega(cl + ct)$.*

We note that the fact that we get an (almost) asymptotically tight bound of $\tilde{\Theta}(cl + ct)$ for the VC Dimension supports our result from Chapter 4 stating that the parameters c, l, t are the right ones to capture the complexity of first order Horn expressions.

From the results in (Maass and Turán, 1992) and in this chapter we conclude that any algorithm that exactly learns the class of closed Horn expressions must make $\tilde{\Omega}(cl + ct)$ membership and equivalence queries. Notice that this is the best possible lower bound that the VC Dimension can provide. This still leaves a gap to our upper bound derived from the learning algorithm in Chapter 5 which is polynomial in $c + t^v$. The main discrepancy is in the exponential dependence in v. In the next chapter we study the *Certificate Size* which is a powerful technique that

also gives lower bounds for the query complexity of learning in our model.

Chapter 7

The Certificate Size

This chapter is a first step towards finding the certificate size of first order Horn expressions. Here, we explore the certificate size of various classes of boolean expressions. We give constructions of polynomial certificates for monotone CNF, unate CNF and Horn CNF. The construction of certificates for the Horn case is based on an analysis of a standardized representation for Horn expressions which we call saturation, and that might be useful in other settings.

Query complexity can be characterized using the combinatorial notion of *certificate size* (Hellerstein et al., 1996; Hegedus, 1995) — see also (Balcázar, Castro, and Guijarro, 1999; Angluin, 2001). In particular, Hellerstein et al. (1996) and Hegedus (1995) show that a class $\mathcal{T}$ is efficiently learnable from equivalence and membership queries if and only if the class $\mathcal{T}$ has polynomial certificates. Informally, $\mathcal{T}$ has polynomial certificates if for every concept not in $\mathcal{T}$ there is a small set of instances in the domain that distinguishes it from all the concepts in $\mathcal{T}$ (we give formal definitions later). Note that only query complexity is considered here and running time is not measured, so this notion is weaker than polynomial time learning with equivalence and membership queries. However, since finding polynomial certificates may be easier than finding efficient algorithms for a particular concept class, certificates make an attractive choice for exploring the learnability question.

The learnability results that follow from our certificate constructions for mono-

tone, unate and Horn CNF are weaker than the learning algorithms for these classes (Valiant, 1984; Angluin, 1988; Bshouty, 1995; Angluin, Frazier, and Pitt, 1992; Frazier and Pitt, 1993) since we obtain query complexity results and the results cited are for time complexity. However, the certificate constructions which we give are different from those implied by these earlier algorithms, and may be useful in suggesting new learning algorithms. We also give new lower bounds on certificate size for each of these concept classes. For some parameter settings, our lower bounds imply that our new certificate constructions are exactly optimal.

Finally, we also consider a natural generalization of these classes, namely the class of renamable Horn CNF expressions. While unate CNF and Horn CNF each have polynomial certificates, we give an exponential lower bound on certificate size for renamable Horn CNF. This answers an open question of Feigelson (1998) and proves that renamable Horn CNF is not efficiently learnable in polynomial time from membership and equivalence queries.

7.1 Definitions and notation

Here we introduce some of the notation and definitions that we use in this chapter.

Definition 30 Let $x, y \in \{0,1\}^n$ be two assignments. Their intersection $x \cap y$ is the assignment that sets to 1 only those variables that are 1 in both x and y.

Definition 31 The *DNF size* of a boolean function $f \subseteq \{0,1\}^n$, denoted $|f|_{DNF}$, is the minimum number of terms in a DNF representation of f. The *CNF size* of f, $|f|_{CNF}$, is defined analogously. In general, let $\mathcal{R}$ be a representation class for boolean formulas. Then $|f|_{\mathcal{R}}$ is the size of a minimal representation for f in $\mathcal{R}$. If $f \notin \mathcal{R}$, we assign $|f|_{\mathcal{R}} = \infty$.

Definition 32 Let $\mathcal{B}$ be a boolean class, i.e. $\mathcal{B} \subseteq 2^{\{0,1\}^n}$. Then $\mathcal{B}^{\leq m}$ denotes the subclass of $\mathcal{B}$ of concepts of size at most m.

Definition 33 Let $\mathcal{R}$ be a class of propositional expressions defining a boolean concept class $\mathcal{B}$. The class $\mathcal{R}$ has *polynomial certificates* if there exist two polynomials $p(\cdot,\cdot)$ and $q(\cdot,\cdot)$ such that for every $n, m > 0$ and for every boolean function $f \subseteq \{0,1\}^n$ s.t. $|f|_{\mathcal{R}} > p(m,n)$, there is a set of assignments $Q \subseteq \{0,1\}^n$ satisfying the following:

1. $|Q| \leq q(m,n)$ and
2. for every $g \in \mathcal{B}^{\leq m}$ there is some $x \in Q$ s.t. $g(x) \neq f(x)$

In other words, (2) states that no function in $\mathcal{B}^{\leq m}$ is consistent with f over Q.

7.2 Certificates for monotone and unate CNFs

In this section we construct polynomial certificates for anti-monotone CNFs and then we generalize the construction to unate DNFs. This is to facilitate the presentation of certificates for Horn CNF. A certificate for unate DNF was given by Feigelson (1998) (formal definition of unate DNF can be found below):

Theorem 63 (Feigelson (1998)) *The classes of monotone and unate functions under DNF have polynomial size certificates with $p(m,n) = m$ and $q(m,n) = O(mn)$.* ■

Feigelson's construction is based on the fact that to show that a unate DNF function has more than m terms, it is sufficient to prove that it has $m+1$ minterms, which can be done by including in the certificate $m+1$ positive assignments corresponding to the minterms and $O(mn)$ negative assignments corresponding to the assignments one level below the positive ones. A term t is a *minterm* for a boolean function f if $t \models f$ but $t' \not\models f$ for every other term $t' \subset t$.

We next show a construction that achieves a certificate of size $O(m^2)$ which improves Feigelson's construction when $m < n$.

An anti-monotone CNF expression is a CNF where all variables appear negated. In this case we have that anti-monotone CNFs satisfy:

$$\forall x, y \in \{0,1\}^n : \text{ if } x < y \text{ then } f(x) \geq f(y),$$

where $<$ between assignments denotes the standard bit-wise relational operator.

Notice that an anti-monotone CNF expression can be seen as a Horn CNF whose clauses have empty consequents. As an example, the anti-monotone CNF $(\bar{a} \vee \bar{b}) \wedge (\bar{b} \vee \bar{c})$ is equivalent to the Horn CNF $(ab \rightarrow \texttt{false}) \wedge (bc \rightarrow \texttt{false})$.

Theorem 64 *The class of anti-monotone CNF has polynomial size certificates with* $p(m,n) = m$ *and* $q(m,n) = \binom{m+1}{2} + m + 1$.

Proof. Fix $m, n > 0$. Fix any $f \subseteq \{0,1\}^n$ s.t. $|f|_{anti-monCNF} > p(m,n) = m$. We proceed by cases.

Case 1. f is not anti-monotone. In this case, there must exist two assignments $x, y \in \{0,1\}^n$ s.t. $x < y$ but $f(x) < f(y)$ (otherwise f would be anti-monotone). Let $Q = \{x, y\}$. Notice that by definition no anti-monotone CNF can be consistent with Q. Moreover, $|Q| = 2 \leq q(m,n)$.

Case 2. f is anti-monotone. Let $c_1 \wedge c_2 \wedge .. \wedge c_m \wedge .. \wedge c_k$ be a minimal representation for f. Notice that $k \geq m+1$ since $|f|_{anti\text{-}monCNF} > p(m,n) = m$. Define assignment $x^{[c_i]}$ as the assignment that sets to 1 exactly those variables that appear in c_i's antecedent. For example, if $n = 5$ and $c_i = v_3 v_5 \rightarrow \texttt{false}$ then $x^{[c_i]} = 00101$.

Remark 2 Notice that every $x^{[c_i]}$ falsifies c_i (antecedent is satisfied but consequent is `false`) but satisfies every other clause in f. If this were not so, then we would have that some other clause c_j in f is falsified by $x^{[c_i]}$, that is, the antecedent of c_j is true and therefore all variables in c_j appear in c_i as well (i.e. $c_j \subseteq c_i$). This is a contradiction since c_i would be redundant and we are looking at a minimal representation of f.

Now, define the set $Q = Q^+ \cup Q^-$ where

$$Q^- = \left\{ x^{[c_i]} \,\middle|\, 1 \leq i \leq m+1 \right\} \text{ and } Q^+ = \left\{ x^{[c_i]} \cap x^{[c_j]} \,\middle|\, 1 \leq i < j \leq m+1 \right\}.$$

Notice that $|Q| \leq \binom{m+1}{2} + m + 1 = q(m,n)$. The assignments in Q^- are negative for f, since $x^{[c_i]}$ clearly falsifies clause c_i (and hence it falsifies f). The assignments in Q^+ are positive for f. To see this, suppose some $x^{[c_i]} \cap x^{[c_j]} \in Q^+$ is negative. Then there is some clause c in f that is falsified by $x^{[c_i]} \cap x^{[c_j]} \in Q^+$. That is, all variables in c are set to 1 by $x^{[c_i]} \cap x^{[c_j]} \in Q^+$. Therefore, all variables in c are set to 1 by $x^{[c_i]}$ and $x^{[c_j]}$ and they falsify the same clause which is a contradiction by the remark above. Hence, all assignments in Q^+ are positive for f.

It is left to show that no anti-monotone CNF g s.t. $|g|_{anti\text{-}monCNF} \leq m$ is consistent with f over Q. Fix any $g = c'_1 \wedge .. \wedge c'_l$ with $l \leq m$. If g is consistent with Q^-, then there is a $c' \in g$ falsified by two different $x^{[c_i]}, x^{[c_j]} \in Q^-$ — we have $m+1$ assignments in Q^- but strictly fewer clauses in g. Since they falsify c', all variables in c' are set to 1 in both $x^{[c_i]}$ and $x^{[c_j]}$. Therefore, all variables in c' are set to 1 in their intersection $x^{[c_i]} \cap x^{[c_j]}$. Hence, clause c' (and therefore g) is falsified by $x^{[c_i]} \cap x^{[c_j]}$. Thus, $x^{[c_i]} \cap x^{[c_j]} \in Q^+$ is negative for g and g and f cannot be consistent. ■

By duality of the boolean operators and DNF/CNF representations we get that monotone CNF, monotone DNF and anti-monotone DNF have polynomial certificates of size $O(min(mn, m^2))$.

Now, we generalize the previous construction to *unate* DNF. First we need some useful definitions.

Definition 34 Let $a, x, y \in \{0,1\}^n$ be three assignments. The inequality between assignments $x \leq_a y$ is defined as $x \oplus a \leq y \oplus a$, where $\leq$ is the bit-wise standard relational operator and $\oplus$ is the bit-wise exclusive OR. Also, we note $x <_a y$ iff $x \leq_a y$ but $y \not\leq_a x$.

Definition 35 A boolean DNF function f (of arity n) is *unate* if there exists some assignment a such that $\forall x, y \in \{0,1\}^n : (x <_a y \implies f(x) \leq f(y))$. Equivalently, a variable cannot appear both negated and unnegated in any minimal DNF representation of f. Variables are either monotone or anti-monotone.

Definition 36 Let $a, x, y \in \{0,1\}^n$ be three assignments. Let $a[i]$ be the i-th value of assignment a. The unate intersection $x \cap_a y$ is defined as:

$$(x \cap_a y)[i] = \begin{cases} x[i] \wedge y[i] & \text{if } a[i] = 0 \\ x[i] \vee y[i] & \text{otherwise} \end{cases}$$

The following is a generalization of Theorem 64; its proof follows along the same lines:

Theorem 65 *Unate DNFs have polynomial size certificates with* $p(m,n) = m$ *and* $q(m,n) = \binom{m+1}{2} + m + 1$.

Proof. Let $p(m,n) = m$ and $q(m,n) = \binom{m+1}{2} + m + 1$. Fix $m, n > 0$. Fix any $f \subseteq \{0,1\}^n$ s.t. $|f|_{unateDNF} > p(m,n) = m$. Now we proceed by cases.

Case 1. f is not unate. In this case, there must exist four assignments $x, y, z, w \in \{0,1\}^n$ and a position i $(1 \leq i \leq n)$ such that:

- $x[j] = y[j]$ for all $1 \leq j \leq n, j \neq i$ and $x[i] < y[i]$
- $z[j] = w[j]$ for all $1 \leq j \leq n, j \neq i$ and $z[i] > w[i]$
- $f(x) > f(y)$ and $f(z) > f(w)$

Let $Q = \{x, y, z, w\}$. Notice that $|Q| \leq q(m,n)$. To see that no unate DNF can be consistent with f over Q, take any unate DNF g and suppose it is. Let a be the assignment mentioned in Definition 35 for g. If $a[i] = 0$ (i-th value given by a) then we have that $x \leq_a y$ but $g(x) > g(y)$. If $a[i] = 1$ then $z \leq_a w$ but $g(z) > g(w)$. Therefore there can not be any unate function consistent with f over Q.

Case 2. f is unate. Let a be the assignment witnessing f being unate. Suppose w.l.o.g. (just rename variables accordingly) that $a = 0^r 1^{n-r}$ where r is the number of monotone variables in f. Suppose that the variables in f are $\{v_1, ..., v_n\}$. Therefore, variables $\{v_1, ..., v_r\}$ appear always positive in f and variables $\{v_{r+1}, ..., v_n\}$ appear always negative. Let $t_1 \vee t_2 \vee ... \vee t_m \vee ... \vee t_k$ be a minimal DNF representation of f. Notice that $k \geq m+1$ since $|f|_{unateDNF} > p(m,n) = m$. Define j-th value of assignment $x^{[t_i]}$ as (for $1 \leq j \leq n$):

$$x^{[t_i]}[j] = \begin{cases} 1 & \text{if } j \leq r \text{ and } v_j \text{ appears in } t_i \\ 0 & \text{if } j \leq r \text{ and } v_j \text{ does not appear in } t_i \\ 0 & \text{if } j > r \text{ and } \bar{v}_j \text{ appears in } t_i \\ 1 & \text{if } j > r \text{ and } \bar{v}_j \text{ does not appear in } t_i \end{cases}$$

Now, define the sets

$$Q^+ = \left\{ x^{[t_i]} \;\middle|\; 1 \leq i \leq m+1 \right\}$$

$$Q^- = \left\{ x^{[t_i]} \cap_a x^{[t_j]} \;\middle|\; 1 \leq i < j \leq m+1 \right\}$$

$$Q = Q^+ \cup Q^-.$$

Notice that $|Q| \leq |Q^-| + |Q^+| \leq \binom{m+1}{2} + m + 1 = q(m,n)$. The assignments in Q^+ are positive for f, since $x^{[t_i]}$ clearly satisfies term t_i. The assignments in Q^- are negative for f. To see this, suppose some $x^{[t_i]} \cap_a x^{[t_j]}$ is positive. Let $t \in f$ be a term that is satisfied by $x^{[t_i]} \cap_a x^{[t_j]}$. That means that variables among $\{v_1, ..., v_r\}$ appearing in t are set to 1 by $x^{[t_i]} \cap_a x^{[t_j]}$ (therefore in $x^{[t_i]}$ and $x^{[t_j]}$, too) and variables among $\{v_{r+1}, ..., v_n\}$ appearing in t are set to 0 by $x^{[t_i]} \cap_a x^{[t_j]}$ (therefore in $x^{[t_i]}$ and $x^{[t_j]}$, too). That is, $t \subseteq t_i$ and $t \subseteq t_j$ which is a contradiction because some term in f would be redundant and we have assumed some minimal representation of f. Hence, all assignments in Q^- are negative for f.

It is left to show that no unate DNF g s.t. $|g|_{unateDNF} \leq m$ is consistent with f over Q. Fix any $g = t'_1 \vee ... \vee t'_l$ with $l \leq m$. If g is consistent with Q^+, then

there is a $t' \in g$ satisfied by two different $x^{[t_i]}, x^{[t_j]} \in Q^+$ (because we have $m+1$ assignments in Q^+ but less than $m+1$ terms in g). So we have that $x^{[t_i]} \models t'$ and $x^{[t_j]} \models t'$. Hence, all variables appearing in t' have the same value in $x^{[t_i]}$ and $x^{[t_j]}$, and therefore they have the same value in $x^{[t_i]} \cap_a x^{[t_j]}$ so that $x^{[t_i]} \cap_a x^{[t_j]} \models t'$. Then, $x^{[t_i]} \cap_a x^{[t_j]} \in Q^-$ is positive for g and g is not consistent with f over Q. ■

Corollary 66 *Unate CNF/DNF have certificates of size* $O(min(mn, m^2))$.

Proof. By duality of the DNF/CNF representations. ■

7.3 Saturated Horn CNFs

This section develops a "standardized" representation for propositional Horn expressions which can be obtained by an operation we call saturation. We establish properties of saturated expressions that make it possible to construct a set of certificates in a similar way to the case of anti-monotone CNF.

Definition 37 Let f be a Horn CNF. We define *Saturation*(f) as the Horn expression returned by the following procedure:

SATURATION(f)

1 $Sat \leftarrow f$
2 **repeat**
3 **if** there exist $s_i \rightarrow b_i$, $s_j \rightarrow b_j$ in Sat s.t. $b_i \neq b_j$, $s_j \subseteq s_i$, $b_j \notin s_i$
4 **then** $s'_i \leftarrow s_i \cup \{b_j\}$
5 replace $s_i \rightarrow b_i$ with $s'_i \rightarrow b_i$ in Sat.
6 **until** no changes are made to Sat
7 **return** Sat

This procedure must terminate within $O(mn)$ time steps, where m is the number of clauses in the initial expression, and n is the number of variables: we can add up

to mn variables to the antecedents of the clauses in f, and after every execution of the loop at least one variable is added.

By a *saturation of* f we mean any of the possible outcomes of the procedure SATURATION(f).

Example 14 Notice that an expression can have many possible saturations. As an example, take $f = \{a \to b, a \to c\}$; this expression has two possible saturations: $Sat_1 = \{ac \to b, a \to c\}$ and $Sat_2 = \{a \to b, ab \to c\}$. Clearly, the result depends on the order in which we saturate clauses.

Lemma 67 *Every Horn expression is logically equivalent to its saturation.*

Proof. We show inductively that after every iteration of the main loop in the procedure above the logical value of the expression being computed does not change. Suppose, then, that we are about to change the expression Sat. Let Sat be the expression before the change and Sat' after. Let $s_i \to b_i \in Sat$ be the clause updated to $s_i' \to b_i \in Sat'$. We have to show that $Sat \equiv Sat'$. The direction $Sat \models Sat'$ is easy. To show this, notice that $s_i \to b_i \models s_i' \to b_i$ since s_i' has just one more variable than s_i. For the other direction $Sat' \models Sat$ fix an arbitrary x such that $x \models Sat'$. We show that $x \models Sat$, too. It holds that $x \models C'$ for all clauses $C' \in Sat'$. Trivially, $x \models C$ for all $C \in Sat$ other than $s_i \to b_i$. It is left to show that $x \models s_i \to b_i$ also. The two following cases arise: (1) the extra variable in s_i' (w.r.t. s_i) is set to 1 by x, or (2) it is set to 0. If (1) holds, then it is easy to see that $x \models s_i \to b_i$ iff $x \models s_i' \to b_i$ and we conclude $x \models s_i \to b_i$. If (2) holds, then let $s_j \to b_j$ be the clause that was used to add the extra variable (b_j) to s_i'. We have seen that $x \models s_j \to b_j$ and that b_j is set to 0, therefore s_j must be falsified by x (that is some variable in s_j is set to 0 by x). Notice, too, that $s_j \subseteq s_i$. Hence, some variable in s_i must be set to 0 by x, too. Thus $x \models s_i \to b_i$ as required. ■

Notice that we use the notion of a "sequential" saturation in the sense that we use the updated expression to continue the process of saturation. There is a notion

of "simultaneous" saturation that uses the original expression to saturate all the clauses. Lemma 67 does not hold for simultaneous saturation. An easy example illustrates this. Let $f = \{a \to b, a \to c\}$. Clearly, $SimSat(f) = \{ac \to b, ab \to c\}$ is not logically equivalent to f (notice $f \models a \to b$ but $SimSat(f) \not\models a \to b$).

Definition 38 An expression f is saturated iff $f = Saturation(f)$.

Definition 39 A clause C in a Horn expression f is *redundant* if $f \setminus \{C\} \equiv f$. An expression f is redundant if it contains a redundant clause.

Lemma 68 *If a Horn expression f is non-redundant, then all of its saturations are non-redundant, too.*

Proof. We show that if any fixed but arbitrary saturation of f is redundant (call it Sat'), then f has to be redundant, too. Assume that Sat' is redundant. We argue inductively on the number of changes made to the expression f during the saturation process.

Base case: f is saturated (i.e. $f = Sat'$). Clearly f is redundant if Sat' is.

Step case: f is not saturated (i.e. $f \neq Sat'$). Consider the last change made by the saturation procedure before obtaining Sat'. Let Sat be the expression just before obtaining Sat'; let $s_i \to b_i \in Sat$ be the clause replaced by $s'_i \to b_i \in Sat'$ using $s_j \to b_j \in Sat$. Notice that $s'_i = s_i \cup \{b_j\}$ and that Sat and Sat' coincide in clauses other than $s_i \to b_i$ and $s'_i \to b_i$. Since Sat' is redundant, there is a clause $C' \in Sat'$ that can be deduced from the other clauses of Sat'. Therefore, there is a minimal derivation graph G' of $Sat' \setminus C' \vdash C'$. Denote $C \in Sat$ the clause corresponding to C' in Sat'. Now we proceed by cases. In every case we transform G' proving redundancy of the clause C in Sat.

Case 1a. If $s'_i \to b_i$ does not appear in the derivation graph and $C' \neq s'_i \to b_i$, then no modification is needed to show that $C = C'$ is redundant in Sat.

Case 1b(i). If $C' = s'_i \to b_i$ and the added b_j does not appear in G', then no modification is needed and G' shows that $C = s_i \to b_i$ is redundant in Sat.

Case 1b(ii). If $C' = s'_i \rightarrow b_i$ and the added b_j appears in G', then we just add edges $b \rightarrow b_j$ for every $b \in s_j$ (first add nodes $b \in s_j$ not in G' already!). Notice that this is a valid derivation graph for the redundant $C = s_i \rightarrow b_i$ from Sat.

Case 2. Now suppose that the updated clause appears in the proof. Notice that the variable b_j has to be different from the consequent of the redundant clause. If this were not so, we would have a smaller derivation graph, contradicting the fact that we assume a minimal one. Therefore, the clause $s_j \rightarrow b_j$ used to saturate cannot be C' itself. We modify G' in the following way. If the variable b_j has only one edge going to b_i, we simply remove b_j, the edge $b_j \rightarrow b_i$, all edges $* \rightarrow b_j$ reaching b_j and any unconnected parts remaining in the derivation graph. If b_j has more edges pointing at variables other than b_i, we remove the edge $b_j \rightarrow b_i$ but add edges $b \rightarrow b_j$ for every $b \in s_j$ (first adding any $b \in s_j$ not in G' already).

In either case, we obtain that $C \in Sat$ is redundant. Applying the induction hypothesis, we conclude that (the possibly unsaturated version of) C is redundant in the initial f. ■

The converse of the previous lemma does not hold. That is, there are redundant expressions f with non-redundant saturations. As an example: $f = \{ab \rightarrow c, c \rightarrow d, ab \rightarrow d\}$ is clearly redundant since the third clause $ab \rightarrow d$ can be deduced from the first two. If we saturate the first clause with the third, we obtain: $Saturation(f) = \{abd \rightarrow c, c \rightarrow d, ab \rightarrow d\}$ which is not redundant! However, if we saturate the third clause with the first, we obtain a redundant saturation $Saturation'(f) = \{ab \rightarrow c, c \rightarrow d, \underline{abc \rightarrow d}\}$.

Lemma 69 *Let f be a non-redundant Horn expression. Let $s_i \rightarrow b$ and $s_j \rightarrow b$ be any two distinct clauses in f with the same consequent. Then, $s_i \not\subseteq s_j$.*

Proof. If $s_i \subseteq s_j$, then $s_i \rightarrow b$ subsumes $s_j \rightarrow b$ and f is redundant. ■

Lemma 70 *Let f be a non-redundant, saturated Horn expression. Let c be any clause in f. Let $x^{[c]}$ be the assignment that sets to one exactly those variables in the*

antecedent of c. Then, $x^{[c]}$ falsifies c but satisfies every other clause c' in f.

Proof. Let $c = s \to b$. Clearly, $x^{[c]}$ falsifies c: its antecedent is satisfied but its consequent is not. It also satisfies every other clause $c' = s' \to b'$ in f. To see this, we look at the following two cases: if $s' \not\subseteq s$, there is a variable in s' not in s. Hence $x^{[c]} \not\models s'$ and $x^{[c]} \models c'$. Otherwise, $s' \subseteq s$ and Lemma 69 guarantees that $b \neq b'$ (otherwise there would be a redundant clause in f). Furthermore, $b' \in s$ (otherwise f would not be saturated). Thus, $x^{[c]} \models b'$ and $x^{[c]} \models c'$. ■

7.4 Certificates for Horn CNF

The following characterization is due to McKinsey (1943), although it was stated in a different context and in more general terms. It was further explored by Horn (1956). Finally, a proof adapted to our setting can be found e.g. in (Khardon and Roth, 1996). Horn CNF expressions are characterized by

$$\forall x, y \in \{0,1\}^n : \text{ if } x \models f \text{ and } y \models f, \text{ then } x \cap y \models f \tag{7.1}$$

Theorem 71 *Horn CNFs have polynomial size certificates with $p(m,n) = m(n+1)$ and $q(m,n) = \binom{m+1}{2} + m + 1$.*

Proof. Fix $m, n > 0$. Fix any $f \subseteq \{0,1\}^n$ s.t. $|f|_{hornCNF} > p(m,n) = m(n+1)$. Again, we proceed by cases.

Case 1. f is not Horn. In this case, there must exist two assignments $x, y \in \{0,1\}^n$ s.t. $x \models f$ and $y \models f$ but $x \cap y \not\models f$ (otherwise f would be Horn). Let $Q = \{x, y, x \cap y\}$. Notice that by (7.1) no Horn CNF can be consistent with Q. Moreover, $|Q| = 3 \leq q(m,n)$.

Case 2. f is Horn. Let $c_1 \wedge c_2 \wedge \ldots \wedge c_{k'}$ be a minimal, saturated representation of f. Notice that $k' \geq m(n+1) + 1$ since $|f|_{hornCNF} > p(m,n) = m(n+1)$ and saturation does not produce redundant clauses when starting from a non-redundant representation (see Lemma 68). Since there are more than $m(n+1)$ clauses, there

must be at least $m+1$ clauses sharing a single consequent in f (there are at most $n+1$ different consequents among the clauses in f — including the constant `false`). Let these clauses be $c_1 = s_1 \rightarrow b, .., c_k = s_k \rightarrow b$, with $k \geq m+1$. Define assignment $x^{[c_i]}$ as the assignment that sets to 1 exactly those variables that appear in c_i's antecedent. For example, if $n = 5$ and $c_i = v_3 v_5 \rightarrow v_2$ then $x^{[c_i]} = 00101$. Define the set $Q = Q^+ \cup Q^-$ where

$$Q^- = \left\{ x^{[c_i]} \;\middle|\; 1 \leq i \leq m+1 \right\} \text{ and } Q^+ = \left\{ x^{[c_i]} \cap x^{[c_j]} \;\middle|\; 1 \leq i < j \leq m+1 \right\}.$$

Notice that $|Q| = |Q^+| + |Q^-| \leq \binom{m+1}{2} + m + 1 = q(m,n)$. The assignments in Q^- are negative for f, since $x^{[c_i]}$ clearly falsifies clause c_i (and hence it falsifies f). The assignments in Q^+ are positive for f. To see this, we show that every assignment in Q^+ satisfies every clause in f. Take any assignment $x^{[c_i]} \cap x^{[c_j]} \in Q^+$. For clauses c with a different consequent than c_i (thus $c \neq c_i, c \neq c_j$), Lemma 70 shows that $x^{[c_i]} \models c$ and $x^{[c_j]} \models c$. Since c is Horn, $x^{[c_i]} \cap x^{[c_j]} \models c$. For clauses with the same consequent as c_i (and c_j), we have two cases. Either $c \neq c_i$ or $c \neq c_j$. If $c \neq c_i$, then Lemma 69 guarantees that $s \not\subseteq s_i$, where s is c's antecedent. Therefore some variable in s is set to 0 by $x^{[c_i]}$ and hence by $x^{[c_i]} \cap x^{[c_j]}$, too. Thus, $x^{[c_i]} \cap x^{[c_j]} \models c$. The other case is analogous. Hence, all assignments in Q^+ are positive for f.

It is left to show that no Horn CNF g s.t. $|g|_{hornCNF} \leq m$ is consistent with f over Q. Fix any $g = c'_1 \wedge .. \wedge c'_l$ with $l \leq m$. If g is consistent with Q^-, then there is a $c' \in g$ falsified by two different $x^{[c_i]}, x^{[c_j]} \in Q^-$ (because we have $m+1$ assignments in Q^- but strictly fewer clauses in g). Since they falsify c', all variables in the antecedent of c' are set to 1 in both $x^{[c_i]}$ and $x^{[c_j]}$. Also, in both assignments the consequent of c' is set to 0. Therefore, the assignment $x^{[c_i]} \cap x^{[c_j]}$ sets all variables in the antecedent of c' to 1 and the consequent to 0, too. Hence, clause c' (and therefore g) is falsified by $x^{[c_i]} \cap x^{[c_j]}$. Thus, $x^{[c_i]} \cap x^{[c_j]} \in Q^+$ is negative for g and g cannot be consistent with f over Q. ■

Next, we include an alternative construction of polynomial certificates for Horn CNF expressions. The reason for this is that this alternative construction, although having worse query complexity bounds, seems better suited to be generalized to first order Horn expressions. We need the following fact.

Fact 72 Let $(S, \preceq)$ be a quasi-order (see page 21). Let r be the length of the longest strict chain $s_1 \prec s_2 \prec .. \prec s_{r-1} \prec s_r$. Then, any subset of S of cardinality at least $r+1$ must contain two elements s, s' such that $s \not\preceq s'$ and $s' \not\preceq s$. ■

Theorem 73 *Horn CNFs have polynomial size certificates with* $p(m,n) = m(n+1)$ *and* $q(m,n) = \binom{m(n+1)+1}{2} + m(n+1) + 1$.

Proof. Fix $m, n > 0$. Fix any $f \subseteq \{0,1\}^n$ s.t. $|f|_{hornCNF} > p(m,n) = m(n+1)$.

Case 1. f is not Horn. Exactly as *Case 1* in the proof of Theorem 71.

Case 2. f is Horn. Let $c_1 \wedge c_2 \wedge .. \wedge c_{k'}$ be a minimal, saturated representation of f. Notice that $k' \geq m(n+1)+1$ since $|f|_{hornCNF} > p(m,n) = m(n+1)$. Define the set $Q = Q' \cup Q^-$ where $Q^- = \left\{ x^{[c_i]} \;\middle|\; 1 \leq i \leq m(n+1)+1 \right\}$ and $Q' = \left\{ x^{[c_i]} \cap x^{[c_j]} \;\middle|\; 1 \leq i < j \leq m(n+1)+1 \right\}$. Notice that $|Q| = |Q'| + |Q^-| \leq \binom{m(n+1)+1}{2} + m(n+1) + 1 = q(m,n)$. In contrast to the proof of Theorem 71, the set Q' may now contain negative and positive assignments: suppose $n = 4$ and f contains three clauses $v_1 v_2 \rightarrow v_3$, $v_4 \rightarrow v_3$ and $v_1 \rightarrow v_2$. Then, $Q^- \supseteq \{1100, 0001, 1000\}$ and $Q' \supseteq \{1000, 0000\}$. Clearly, 1000 is negative and 0000 is positive.

Fix an arbitrary Horn CNF $g = c'_1 \wedge .. \wedge c'_l$ with $l \leq m$. Suppose g is consistent with f over Q. Suppose that each clause in g is falsified only by $n+1$ assignments in Q^-. Then only $m(n+1)$ assignments in Q^- are negative for g, and hence g and f cannot be consistent over Q^-. Hence, there exists a clause c' in g that is falsified by at least $n+2$ assignments in Q^-. Let $n+2$ of these assignments be $A = \{x^{[c^*_1]}, \ldots, x^{[c^*_{n+2}]}\}$, and each $c^*_l = s^*_l \rightarrow b^*_l$ for all $1 \leq l \leq n+2$. Now we consider chains $s^*_{l_1} \subset .. \subset s^*_{l_p}$, where $1 \leq l_j \leq n+2$ for all $1 \leq j \leq p$ (i.e. we consider proper chains under set inclusion among the antecedents of the clauses corresponding to the assignments in A). Clearly, any such chain is of length at most $n+1$ because

there are n variables and $s^*_{l_1}$ might be empty. We apply Fact 72 and conclude that there are two assignments $x^{[c^*_i]}, x^{[c^*_j]}$ in $A \subseteq Q^-$ such that $s^*_i \not\subseteq s^*_j$ and $s^*_j \not\subseteq s^*_i$.

It holds that $x^{[c^*_i]} \not\models c'$ and $x^{[c^*_j]} \not\models c'$, and we conclude that $x^{[c^*_i]} \cap x^{[c^*_j]} \not\models c'$ and hence $x^{[c^*_i]} \cap x^{[c^*_j]} \not\models g$ (following the same argument as in the proof of Theorem 71).

Now we show that $x^{[c^*_i]} \cap x^{[c^*_j]} \models c$ for every clause $c \in f$. For clauses c other than c^*_i and c^*_j, Lemma 70 guarantees that $x^{[c^*_i]} \models c$ and $x^{[c^*_j]} \models c$. Since c is Horn, $x^{[c^*_i]} \cap x^{[c^*_j]} \models c$. For $c = c^*_i$, it holds that $x^{[c^*_i]} \cap x^{[c^*_j]} \models c^*_i$ because $s^*_i \not\subseteq s^*_j$ implies that $x^{[c^*_i]} \cap x^{[c^*_j]} \not\models s^*_i$ so that $x^{[c^*_i]} \cap x^{[c^*_j]} \models c^*_i$. Similarly, $x^{[c^*_i]} \cap x^{[c^*_j]} \models c^*_j$. We conclude that $x^{[c^*_i]} \cap x^{[c^*_j]} \models f$ since $x^{[c^*_i]} \cap x^{[c^*_j]}$ satisfies every clause in f. Thus, f and g disagree on $x^{[c^*_i]} \cap x^{[c^*_j]} \in Q'$, which is a contradiction. ■

7.5 Learning from entailment

In the model of learning from interpretations, a certificate is a set of interpretations (or assignments in the case of propositional logic). So far we have showed how to construct interpretation certificates, since our constructions were based on assignments. These constructions provide bounds for the complexity of learning classes in the model of learning from interpretations. In the model of learning from entailment, a certificate is a set of clauses. We present a general transformation that allows us to obtain an entailment certificate from an interpretation certificate. Similar observations have been made before in different context e.g. (Khardon and Roth, 1999; De Raedt, 1997) where one transforms efficient algorithms instead of just certificates. Note however, that for computational efficiency we must be able to solve the implication problem for the language of hypotheses used by the algorithm.

Definition 40 Let x be an interpretation. Then $ones(x)$ is the set of variables that are set in x.

Lemma 74 *Let f be a boolean expression and x an interpretation. Then,*

$$x \models f \text{ if and only if } f \not\models ones(x) \rightarrow \bigvee_{b \notin ones(x)} b.$$

Proof. Suppose $x \models f$. By construction, $x \not\models ones(x) \rightarrow \bigvee_{b \notin ones(x)} b$. Suppose by way of contradiction that $f \models ones(x) \rightarrow \bigvee_{b \notin ones(x)} b$. But since $x \not\models ones(x) \rightarrow \bigvee_{b \notin ones(x)} b$ we conclude that $x \not\models f$, which contradicts our initial assumption. Now, suppose $x \not\models f$. Hence, there is a clause $s \rightarrow \bigvee_i b_i$ in f falsified by x. This can happen only if $s \subseteq ones(x)$ and $b_i \notin ones(x)$ for all i. Clearly, $s \rightarrow \bigvee_i b_i \models ones(x) \rightarrow \bigvee_{b \notin ones(x)} b$. Therefore $f \models ones(x) \rightarrow \bigvee_{b \notin ones(x)} b$. ■

Theorem 75 *Let S be an interpretation certificate for a boolean expression f w.r.t. a class $\mathcal{B}$ of boolean expressions. Then, the set $\{ones(x) \rightarrow \bigvee_{b \notin ones(x)} b \mid x \in S\}$ is an entailment certificate for f w.r.t. $\mathcal{B}$.*

Proof. If S is an interpretation certificate for f w.r.t. some class $\mathcal{B}$ of propositional expressions, then for all $g \in \mathcal{B}$ there is some assignment $x \in S$ such that $x \models f$ and $x \not\models g$ or vice versa. Therefore, by Lemma 74, it follows that $f \not\models ones(x) \rightarrow \bigvee_{b \notin ones(x)} b$ and $g \models ones(x) \rightarrow \bigvee_{b \notin ones(x)} b$ or vice versa. Given the arbitrary nature of g the theorem follows. Moreover, both sets have the same cardinality. ■

7.6 Certificate size lower bounds

The certificate results above imply that unate and Horn CNF are learnable with a polynomial number of queries but as mentioned above this was already known. It is therefore useful to review the relationship between the certificate size of a class and its query complexity. From (Hegedus, 1995; Hellerstein et al., 1996) we know that if $CS(\mathcal{B})$ is the certificate size of a certain class $\mathcal{B}$, then its query complexity (denoted $QC(\mathcal{B})$) satisfies:

$$CS(\mathcal{B}) \leq QC(\mathcal{B}) \leq CS(\mathcal{B}) \log(|\mathcal{B}|)$$

For the class of monotone DNF there is an algorithm that achieves query complexity $O(mn)$ (Valiant, 1984; Angluin, 1988). Since $\log(\left|monotoneDNF^{\leq m}\right|) =$

$\Theta(mn)$, a certificate result is not likely to improve the known learning complexity. In the case of Horn CNF, there is an algorithm that achieves query complexity $O(m^2n)$ (Angluin, Frazier, and Pitt, 1992). Since again $\log(\left|hornCNF^{\leq m}\right|) = \Theta(mn)$ improving on known complexity would require a certificate for Horn of size $o(m)$. The results in this section show that this is not possible and in fact that our certificate constructions are optimal. We do this by giving lower bounds on certificate size. Naturally, these also imply lower bounds for the learning complexity.

In particular, for every m, n with $m < n$ we construct an n-variable monotone DNF f of size greater than m and show that any certificate that f has more than m terms must have cardinality at least $q(m,n) = m+1+\binom{m+1}{2}$. We also show that if $m > n$ then there is a monotone DNF of size greater than m that requires a certificate of size $\Omega(mn)$. These results also apply to both unate and Horn CNF/DNF as described below. We first give the result for $m < n$:

Theorem 76 *Any certificate construction for monotone DNF for $m < n$ with $p(m,n) = m$ has size $q(m,n) \geq m+1+\binom{m+1}{2}$.*

Proof. Let $X_n = \{x_1, .., x_n\}$ be the set of n variables and let $m < n$. Let $f = t_1 \vee \cdots \vee t_{m+1}$ where t_i is the term containing all variables (unnegated) except x_i. Such a representation is minimal and hence f has size exactly $m+1$. We show that any set with fewer than $m+1+\binom{m+1}{2}$ assignments cannot certify that f has more than m terms. That is, for any set Q of size less than $m+1+\binom{m+1}{2}$ assignments, we show that there is a monotone DNF with at most m terms consistent with f over Q.

If Q contains at most m positive assignments of weight $n-1$ then it easy to see that the function with minterms corresponding to these positive assignments is consistent with f over Q. Hence we may assume that Q contains at least $m+1$ positive assignments of weight $n-1$. Since f only has $m+2$ positive assignments, one of which is 1^n, Q must include all $m+1$ positive assignments corresponding to the minterms of f. Thus if $|Q| < m+1+\binom{m+1}{2}$ then Q must contain strictly

less than $\binom{m+1}{2}$ negative assignments. Notice that all the intersections between pairs of positive assignments of weight $n-1$ are different and there are $\binom{m+1}{2}$ such intersections. It follows that Q must be missing some intersection between some pair of positive assignments in Q. But then there is an m-term monotone DNF consistent with Q which uses one term for the missing intersection and $m-1$ terms for the other $m-1$ positive assignments. ■

We can strengthen the previous theorem so that for every n a fixed function f serves for all $m < n$. The motivation behind this is that the lower bound in Theorem 76 implies a lower bound on the query complexity of any strongly proper learning algorithm (Hellerstein and Raghavan, 2002; Pillaipakkamnatt and Raghavan, 1996). Such algorithms are only allowed to output hypotheses that are of size at most that of the target expression; this is in contrast with the usual scenario in which learning algorithms are allowed to present hypotheses of size polynomial in the size of the target. In the following certificate lower bound we use a function f of DNF size n, so the resulting lower bound for learning algorithms applies to algorithms which may use hypotheses of size at most $n-1$ (even if the target function is much smaller).

Theorem 77 *Any certificate construction for monotone DNF for $m < n$ with $p(m,n) < n$ has size $q(m,n) \geq m + 1 + \binom{m+1}{2}$.*

Proof. Let $q(m,n) = m + 1 + \binom{m+1}{2}$ and let f be defined as $f = \bigvee_{i \in \{1,..,n\}} t_i$ where t_i is the term containing all variables (unnegated) except x_i. Clearly, all t_i are minterms, f has size exactly n and f is monotone. We show that for any $m < n$ and any set of assignments Q of cardinality strictly less than $q(m,n)$, there is a monotone function g of at most m terms consistent with f over Q.

We first claim that w.l.o.g. we can assume that all the assignments in the potential certificate Q have exactly one bit set to zero (positive assignments) or two bits set to zero (negative assignments). We prove that if Q contains the positive assignment 1^n, or a negative assignment with more than 2 bits set to zero, then we

can replace these by appropriate assignments with exactly 1 or 2 zeros so that any monotone function g consistent with the latter set of assignments (call it Q') is also consistent with Q. Suppose first that we have a function g consistent with f over Q' where the positive assignment $b \in Q$ with all its bits set to 1 has been changed to b' with just one bit set to 0 (choose it arbitrarily). Since g is monotone, g is consistent with f over Q', $b' \leq b$, and $g(b') = 1$, it follows that $g(b) = 1$ and hence g is also consistent with f over the initial Q. Now suppose that we have a function g consistent with the set Q where one negative assignment a with more than two bits set to zero has been (arbitrarily) changed so that some of the extra zero bits are set to one (call the new assignment a'). Since g is consistent with Q', $g(a') = 0$, and since g is also monotone and $a \leq a'$ it follows that $g(a) = 0$, too. Hence, g is consistent with Q in this second case. By induction, our assumption results in no loss of generality.

We may assume, then, that Q is a set of fewer than $q(m, n)$ assignments each of which has either 1 or 2 zeros. We model the problem of finding a suitable monotone function as a graph coloring problem. We map Q into a graph $G_Q = (V, E)$ where $V = \{p \in Q \mid f(p) = 1\}$ and $E = \{(p_1, p_2) \mid \{p_1, p_2, p_1 \cap p_2\} \subseteq Q\}$. Let $|V| = v$ and $|E| = e$.

First we show that if G_Q is m-colorable then there is a monotone function g of DNF size at most m that is consistent with f over Q. It is sufficient that for each color we find a single term t_c that (1) is satisfied by the positive assignments in Q that have been assigned some color c, with the additional condition that (2) t_c is not satisfied by any of the negative assignments in Q. We define t_c as the minterm corresponding to the intersection of all the assignments colored c by the m-coloring. Property (1) is clearly satisfied, since no variable set to zero in any of the assignments is present in t_c. To see that (2) holds it suffices to notice that the assignments colored c form an independent set in G_Q and therefore none of their pair-wise intersections is in Q. By the assumption no negative point below the intersections is in Q either. The resulting consistent function g contains all

minterms t_c. Since the graph is m-colorable, g has at most m terms.

It remains to show that G_Q is m-colorable. Note that the condition $|Q| < q(m,n)$ translates into $v + e < q(m,n)$ in G_Q. If $v \leq m$ then there is a trivial m-coloring. For $v \geq m+1$, it suffices to prove the following: any v-node graph with $v \geq m+1$ with at most $\binom{m+1}{2} + m - v$ edges is m colorable. We prove this by induction on v.

The base case is $v = m+1$; in this case since the graph has at most $\binom{m+1}{2} - 1$ edges it can be colored with only m colors (reuse one color for the missing edge). For the inductive step, note that any v-node graph which has at most $\binom{m+1}{2} + m - v$ edges must have some node with fewer than m neighbors (otherwise there would be at least $vm/2$ nodes in the graph, and this is more than $\binom{m+1}{2} + m - v$ since v is at least $m+2$ in the inductive step). By the induction hypothesis there is an m-coloring of the $(v-1)$-node graph obtained by removing this node of minimum degree and its incident edges. But since the degree of this node was less than m in G, we can color G using at most m colors. ■

Finally, we give an $\Omega(mn)$ lower bound on certificate size for monotone DNF for the case $m > n$. Like Theorem 76 this result gives a lower bound on query complexity for any strongly proper learning algorithm.

Theorem 78 *Any certificate construction for monotone DNF for $m > n$ with $p(m,n) = m$ has size $q(m,n) = \Omega(mn)$.*

Proof. Fix any constant k. We show that for all n and for all $m = \binom{n}{k} - 1$, there is a function f of monotone DNF size $m+1$ such that any certificate showing that f has more than m terms must contain $\Omega(nm)$ assignments.

Fix n, fix k. We define f as the function whose satisfying assignments have at least $n-k$ bits set to 1. Notice that the size of f is exactly $\binom{n}{k} = m+1$. Let P be the set of assignments corresponding to the minterms of f, i.e. P consists of all assignments that have exactly $n-k$ bits set to 1. Let N be the set of assignments that have exactly $n-(k+1)$ bits set to 1. Notice that f is positive for the assignments in P but negative for those in N. Clearly, assignments in P

are minimal weight positive assignments and assignments in N are maximal weight negative assignments. As in the previous proof, we may assume w.l.o.g. that any certificate Q contains assignments in $P \cup N$ only. Notice, too, that $|P| = \binom{n}{k}$ and $|N| = \frac{(m+1)(n-k)}{k+1} = \binom{n}{k+1} = \Omega(mn)$ for constant k. Moreover, any assignment in N is the intersection of two assignments in P.

Let $Q \subseteq P \cup N$. If Q has at most m positive assignments then it is easy to construct a function consistent with Q regardless of how negative examples are placed. Otherwise, Q contains all the $m+1$ positive assignments in P and the rest are assignments in N. If Q misses any assignment in N then we build a consistent function as follows: use the minterm corresponding to the missing intersection to "cover" two of the positive assignments with just one term. The remaining $m-1$ positive assignments in P are covered by one minterm each. Hence, any certificate Q must contain $P \cup N$ and thus is of size $\Omega(nm)$. ■

We close by observing that all of the lower bounds above apply to unate or Horn CNF/DNF as well. This follows from the fact that monotone CNF/DNF is a special case of unate or Horn CNF/DNF and that the function f is outside the class (has size more than m in all cases).

7.7 An exponential lower bound for renamable Horn

In this section we show that renamable Horn CNF expressions do not have polynomial certificates. This answers an open question posed in (Feigelson, 1998) and implies that the class of renamable Horn CNF is not exactly learnable using a polynomial number of membership and equivalence queries.

Definition 41 A boolean CNF function f (of arity n) is *renamable Horn* if there exists some assignment c such that f_c is Horn, where $f_c(x) = f(x \oplus c)$ for all

$x \in \{0,1\}^n$. In other words, the function obtained by renaming the variables according to c is Horn. We call such an assignment c an orientation for f.

To show non-existence of certificates, we need to prove the negation of the property in Definition 33, namely: for all two-variable polynomials $p(\cdot,\cdot)$ and $q(\cdot,\cdot)$ there exist $n, m > 0$ and a boolean function $\hat{f} \subseteq \{0,1\}^n$ s.t. $\left|\hat{f}\right|_{ren\mathcal{H}} > p(m,n)$ such that for every $Q \subseteq \{0,1\}^n$ it holds (1) $|Q| > q(m,n)$ or (2) some $g \in \mathcal{B}^{\leq m}$ is consistent with f over Q.

In particular, we define an $\hat{f}$ that is not renamable Horn, so that $\left|\hat{f}\right|_{ren\mathcal{H}} = \infty > p(m,n)$ holds for any function $p(m,n)$.

Hence, we need to show: for every polynomial $q(\cdot,\cdot)$, there exist $n, m > 0$ and a non-renamable Horn $\hat{f} \subseteq \{0,1\}^n$ s.t. if no $g \in \mathcal{B}^{\leq m}$ is consistent with $\hat{f}$ over some set of assignments Q, then $|Q| > q(m,n)$. We say that a set Q such that no $g \in \mathcal{B}^{\leq m}$ is consistent with $\hat{f}$ over Q is a *certificate that $\hat{f}$ is not small renamable Horn.*

What we actually show is: for each n which is a multiple of 3, there exists a non-renamable Horn $\hat{f} \subseteq \{0,1\}^n$ s.t. if no $g \in \mathcal{B}_{n^6}$ is consistent with $\hat{f}$ over some set of assignments Q, then $|Q| \geq \frac{1}{3}2^{2n/3}$. Equivalently, for every such n every certificate Q that $\hat{f}$ is not a renamable Horn CNF function of size n^6 has to be of super-polynomial (in fact exponential) size. This is clearly sufficient to prove the non-existence of polynomial certificates for renamable Horn boolean functions.

The following lemma due to Feigelson is useful:

Lemma 79 (Feigelson (1998)) *Let f be a renamable Horn function. Then there is an orientation c for f such that $c \models f$.*

Proof. The proof which we include for completeness is due to Feigelson (1998). Let c' an orientation of f such that $c' \not\models f$. Let c be the positive assignment of f which is minimal with respect to the partial order $<_{c'}$. Such an assignment is unique: if a and b are both positive assignments unrelated in the partial order, then $c'' = a \cap_{c'} b$ is positive and $c'' <_{c'} a, b$.

We claim that c is a orientation for f. It suffices to show $a \cap_{c'} b = a \cap_c b$ for all positive assignments a and b. We show that $(a \cap_{c'} b)[i] = (a \cap_c b)[i]$ for all i s.t. $1 \leq i \leq n$. If i is such that $c[i] = c'[i]$ then clearly $(a \cap_{c'} b)[i] = (a \cap_c b)[i]$. Let i be such that $c[i] \neq c'[i]$. Then every positive assignment sets the bit i like $c[i]$: if $a[i] \neq c[i]$ then $(a \cap_{c'} c)[i] = c'[i]$ and thus $(a \cap_{c'} c) <_{c'} c$ (strictly), contradicting the minimality of c. Thus $a[i] = b[i] = c[i]$ and $(a \wedge b)[i] = (a \vee b)[i]$, and therefore $(a \cap_c b)[i] = (a \cap_{c'} b)[i]$. ■

Definition 42 The function $\hat{f}$ which we use is as follows: Let $n = 3k$ for some $k \geq 1$. We define $\hat{f} : \{0,1\}^n \to \{0,1\}$ to be the function whose only satisfying assignments are $0^k1^k1^k, 1^k0^k1^k$, and $1^k1^k0^k$.

Lemma 80 *The function $\hat{f}$ defined above is not renamable Horn.*

Proof. To see that a function f is not renamable Horn with orientation c it suffices to find a triple (p_1, p_2, q) such that $p_1 \models f$, $p_2 \models f$ but $q \not\models f$ where $q = p_1 \cap_c p_2$. By Lemma 79 it is sufficient to check that the three positive assignments are not valid orientations for f:
The triple $(1^k1^k0^k, 1^k0^k1^k, 1^k1^k1^k)$ rejects $c = 0^k1^k1^k$.
The triple $(0^k1^k1^k, 1^k1^k0^k, 1^k1^k1^k)$ rejects $c = 1^k0^k1^k$.
The triple $(0^k1^k1^k, 1^k0^k1^k, 1^k1^k1^k)$ rejects $c = 1^k1^k0^k$. ■

The following lemma is an extension of Lemma 57 from (Feigelson, 1998). We say that a triple (p_1, p_2, q) such that $p_1 \models f$, $p_2 \models f$ but $q \not\models f$ is *suitable* for c if $q \leq_c p_1 \cap_c p_2$.

Lemma 81 *If Q is a certificate that $\hat{f}$ is not small renamable Horn with orientation c, then Q includes a suitable triple (p_1, p_2, q) for c.*

Proof. Suppose that a certificate Q that $\hat{f}$ is not small renamable Horn with orientation c does not include a suitable triple (p_1, p_2, q) for c. That is, $p_1 \models \hat{f}$,

$p_2 \models \hat{f}$ but $q \not\models \hat{f}$ where $q \leq_c p_1 \cap_c p_2$. Feigelson (1998) defines a function g that is consistent with $\hat{f}$ on Q as follows:

$$g(x) = \begin{cases} 1 & \text{if } x \in Q \text{ and } x \models \hat{f} \\ 1 & \text{if } x \leq_c (s_1 \cap_c s_2) \text{ for any } s_1, s_2 \in Q \text{ s.t. } s_1 \models \hat{f} \text{ and } s_2 \models \hat{f} \\ 0 & \text{otherwise.} \end{cases}$$

The function g is consistent with Q since by assumption no negative example is covered by the second condition. Feigelson (1998) shows that:

Claim 82 *The function g is renamable Horn with orientation c.*

Proof. This proof is due to Feigelson (1998); we include it here for completeness. Consider any assignments p_1, p_2 that are positive for g, i.e., $p_1 \models g$ and $p_2 \models g$, and let $t = p_1 \cap_c p_2$. If p_1, p_2 are included in Q, then clearly $t \models g$ by the definition of g. If $p_1 \notin Q$ then $p_1 \leq_c (s_1 \cap_c s_2)$ for some positive $s_1, s_2 \in Q$ (second condition in the definition of g). Since $t \leq_c p_1 \leq_c (s_1 \cap_c s_2)$, then by the definition of g, $t \models g$ as well. The same reasoning applies for the remaining case $p_2 \notin Q$. Hence, g is renamable Horn with orientation c. ■

Now, we show that g is also *small*. We use the fact that our particular $\hat{f}$ is designed to have very few positive assignments. First notice that g only depends on the positive assignments in Q. Moreover, these must be positive assignments for $\hat{f}$. Suppose that Q contains any $l \leq 3$ of these positive assignments. Let these be $x_1, \ldots, x_l$. A DNF representation for g is:

$$g = \bigvee_{1 \leq i \leq l} t_i \vee \bigvee_{1 \leq i < j \leq l} t_{i,j}$$

where t_i is the term that is true for the assignment x_i only and $t_{i,j}$ is the term that is true for the assignment $x_i \cap_c x_j$ and all assignments below it (w.r.t. c). Notice that we can represent this with just one term by removing literals that correspond to maximal values (w.r.t. c).

Since $l \leq 3$, g has at most $3 + \binom{3}{2} = 6$ terms. Hence, g has CNF size at most n^6 (multiply out all terms to get the clauses). Now we use the fact that if there is a CNF formula representing g of size at most n^6, then there must be a (syntactically) renamable Horn representation $\tilde{g}$ for g which is also of size at most n^6: it is well known that if a function h is Horn and g is a non-Horn CNF representation for h, then every clause in g can be replaced with a Horn clause which uses a subset of its literals; see e.g. (McKinsey, 1943) or Claim 6.3 in (Khardon and Roth, 1996). We arrive at a contradiction: Q is not a certificate that $\hat{f}$ is not small renamable Horn with orientation c since $\tilde{g}$ is not rejected. ■

Theorem 83 *For all $n = 3k$, there is a function $\hat{f} : \{0,1\}^n \to \{0,1\}$ which is not renamable Horn such that any certificate Q showing that the renamable Horn size of $\hat{f}$ is more than n^6 must have $|Q| \geq \frac{1}{3}2^{2n/3}$.*

Proof. The Hamming distance between any two positive assignments for $\hat{f}$ is $2n/3$. Since (as observed by Feigelson) the intersection of two different bits equals the minimum of the two bits, any triple can be suitable for at most $2^{n/3}$ orientations. A negative example in Q can appear in at most 3 triples (only 3 choices for p_1, p_2), and hence any negative example in Q contributes to at most $3 \cdot 2^{n/3}$ orientations. The theorem follows since we need to reject all orientations. ■

Corollary 84 *Renamable Horn CNFs do not have polynomial size certificates.*

We conclude by summarizing all the results obtained in the following table:

Class	*LowerBound*		*UpperBound*	
unate DNF/CNF $m < n$	$\binom{m+1}{2} + m + 1$	(Th. 77)	$\binom{m+1}{2} + m + 1$	(Th. 65)
unate DNF/CNF $m \geq n$	$\Omega(mn)^*$	(Th. 78)	$O(mn)$	(Feigelson, 1998)
Horn CNF $m < n$	$\binom{m+1}{2} + m + 1$	(Th. 77)	$\binom{m+1}{2} + m + 1$	(Th. 71)
Horn CNF $m \geq n$	$\Omega(mn)^*$	(Th. 78)	$\binom{m+1}{2} + m + 1$	(Th. 71)
renamable Horn CNF	$\frac{1}{3}2^{2n/3}$	(Th. 83)		

* Strong certificate size only.

Chapter 8

The Subsumption Lattice and Learnability

This chapter is a result of an attempt to generalize the constructions of the certificates for propositional Horn expressions to first order logic. In particular, we have tried to generalize Theorem 73 which uses the fact that propositional clauses have short proper subsumption chains. We show in Section 8.1 that this is not the case in first order logic, which, to the best of our knowledge, was unknown. As proved in Section 8.2, this implies that learning first order Horn clauses is hard if only membership queries are available. Finally, Section 8.3 studies the number of distinct pairings that two clauses can have, showing that it can indeed be exponential in the number of variables used by the clauses. This implies that our learning algorithm of Chapter 5 can make an exponential number of queries in the worst case.

8.1 On the length of proper chains

In this section we study the length of proper subsumption chains of clauses

$$c_1 \prec c_2 \prec .. \prec c_n$$

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 exponential length exist, even if clauses are function free. This implies that simple algorithmic approaches that rely on repeated minimal size subsumption step refinements may require a long time to converge (Nienhuys-Cheng and De Wolf, 1997).

8.1.1 Fully inequated clauses have short proper chains

For reasons that will become clear in the proof of Lemma 88, we use the biased function *WTerms* (see Chapter 4) which counts the number of terms in an expression, with functional terms contributing twice as much as variables. As an example, $WTerms(p(x, f(x), a)) = 5$ whereas $NTerms(p(x, f(x), a)) = 3$.

Lemma 85 *Let c_1, c_2 be two non-trivial, fully inequated clauses. If $c_1 \preceq c_2$, then it must be via a non-unifying substitution (w.r.t. c_1).*

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 hence it cannot be included in any non-trivial clause, contradicting the fact that $c_1 \cdot \theta \subseteq c_2$. ■

Lemma 86 *Let c_1, c_2 be two fully inequated clauses. If $c_1 \preceq c_2$, then*

$$NTerms(c_1) \leq NTerms(c_2).$$

Proof. All distinct terms in c_1 remain distinct in $c_1 \cdot \theta$ because θ is non-unifying by Lemma 85. 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 the lemma follows. ■

Lemma 87 *Let c_1, c_2 be two fully inequated clauses s.t. $c_1 \preceq c_2$. Then,*

$$NLiterals(c_1) \leq NLiterals(c_2).$$

Proof. 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 Lemma 85. ■

Lemma 88 *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 assumption, c_1 and c_2 are such that $c_1 \preceq c_2$ but $c_2 \not\preceq c_1$. Lemmas 86 and 87 guarantee that $NLiterals(c_1) \leq NLiterals(c_2)$ and $WTerms(c_1) \leq WTerms(c_2)$. We 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 $c_1\theta \subseteq c_2$. By Lemma 85, θ is non-unifying w.r.t. c_1. It must be a variable renaming also, since otherwise we would have that $WTerms(c_1) < WTerms(c_2)$. 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 $c_2 \not\preceq c_1$. ■

Lemma 89 *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 88, after each step in the chain (from left to right), either we increase the number of literals, or the quantity *WTerms* increases. By Lemmas 86 and 87, 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 $2t + l$. ■

8.1.2 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 first show that if the maximal arity of a predicate symbol is a, we can produce chains of length $\Omega(a^l)$ with clauses using a distinct variables and at most l literals, where $l \leq a/2$. We then strengthen this result and show that even if we restrict the signature to contain predicate symbols of arity at most 3, chains of exponential length still exist.

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 clause $d_i = p_1, p_2, .., p_k$, we define the next clause d_{i+1} as follows:

1. if p_1 contains 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 15 Suppose p has arity 4 and that $l = 3$. The construction described above produces the following chain of length 11:

$$
\begin{aligned}
& p(z,z,z,z) \\
\succ \quad & p(x_1,z,z,z), p(z,x_1,z,z), p(z,z,x_1,z) \\
\succ \quad & p(x_1,x_2,z,z), p(z,x_1,z,z), p(z,z,x_1,z) \\
\succ \quad & p(z,x_1,z,z), p(z,z,x_1,z) \\
\succ \quad & p(x_2,x_1,z,z), p(z,x_1,x_2,z), p(z,z,x_1,z) \\
\succ \quad & p(z,x_1,x_2,z), p(z,z,x_1,z) \\
\succ \quad & p(z,z,x_1,z) \\
\succ \quad & p(x_2,z,x_1,z), p(z,x_2,x_1,z), p(z,z,x_1,x_2)
\end{aligned}
$$

$$\succ \quad p(z, x_2, x_1, z), p(z, z, x_1, x_2)$$

$$\succ \quad p(z, z, x_1, x_2)$$

$$\succ \quad \emptyset$$

To see that this process always terminates, it is sufficient to observe that we drop atoms that contain a small number of occurrences of the variable z, and every time we replace an existing atom by new ones, the new ones have strictly fewer occurrences of the variable z. Hence, this process terminates in a finite number of steps and the last clause is $c_n = \emptyset$.

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:

$$N(2, s) = 1, \text{ for all } s \geq 0$$

When there are only 2 occurrences of the variable z, the only possible step is to remove the atom, thus obtaining the empty clause. After this, no more generalizations are possible.

$$N(c, 0) = c - 1, \text{ for all } c \geq 2$$

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.

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

This recurrence is obtained by observing that the initial clause containing our single atom can be replaced by $max(0, s-c+1)$ "copies" in a first generalization step. 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 0 "positions" to use for its expansion and is generalized $N(c-1,0)$ times until it is finally dropped; the next atom has 1 "position" to expand since the left-most atom has been dropped, and hence it produces $N(c-1,1)$ generalization steps until it is finally dropped, and so on.

Lemma 90 *$N(c,s) \geq \binom{c}{s+1} - 1$ for $c \geq 2$.*

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

$$
\begin{aligned}
N(c,s) &= 1 + \sum_{i=max(0,s-c+1)}^{s} N(c-1,i) && (8.1)\\
&\geq 1 + N(c-1,s) + N(c-1,s-1) && (8.2)\\
&\geq 1 + \binom{c-1}{s+1} - 1 + \binom{c-1}{s} - 1 && (8.3)\\
&= \binom{c}{s+1} - 1 && (8.4)
\end{aligned}
$$

For (8.2), 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 (8.3) we apply the hypothesis of induction, and for (8.4) 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.

Lemma 91 *Let $Vars(p)$ be the variables occurring in the atom p. For all $d_i = p_1, .., p_k$ 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 $j = 1, .., k-1$.*

Proof. Proved by induction on the updates of d_i. ■

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 $B_1, B_2, .., B_m$ 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 92 *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). We only have to check that the claim is true for the new block $B = p'_1, .., p'_{k'}$ since the hypothesis of induction guarantees that the lemma holds in the rest of the blocks. 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 $p_1 \cdot \theta = p_2$, 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 93 *Let d_i be any clause 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} 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 hypothesis of induction 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 hypothesis of induction applies and we conclude that some variable in B_{i_2} satisfies the property stated in the lemma. 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 94 *Fix some clause $d_i = p_1, .., p_k$ with at least 2 atoms (i.e., $k \geq 2$). Then $p_2 \cdot \theta, .., p_k \cdot \theta \in d_i$ 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. 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 next argue inductively over the blocks' indices starting with m. Consider any atom p in the last block B_m. The fact that $p \cdot \theta \in d_i$ implies by our previous argument that $p \cdot \theta \in B_m$. But since $p \in B_m$, Lemma 92 shows that θ cannot change variables in B_m. Now, fix some block B_j where $j < m$ and assume that variables in blocks $B_{j+1}, .., B_m$ are not changed by θ. Let p be any atom in B_j, and fix some other block $B_{j'}$ s.t. $j < j'$. Lemma 93 guarantees that there exists some variable $x \in \mathit{Vars}(B_{j'})$ that appears in a position in p in which atoms in $B_{j'}$ contain the variable z. Since x is not changed by θ, it cannot be that $p \cdot \theta \in B_{j'}$. $B_{j'}$ is arbitrary among $B_{j+1}, .., B_m$ and therefore $p \cdot \theta \notin B_{j+1}, .., B_m$. The only possibility then is that $p \cdot \theta \in B_j$ in which case Lemma 92 guarantees that the variables in B_j are not changed by θ. This induction shows that θ cannot change variables that appear in the leftmost block of $p_2, .., p_k$, and hence in p_2 as required. ■

Finally, we prove:

Lemma 95 *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 $\emptyset \neq d_i \cdot \theta \subseteq \emptyset = d_{i+1}$. Therefore, $d_{i+1} \neq \emptyset$ and d_i contains at least 2 atoms. $d_i \cdot \theta \subseteq d_{i+1}$ implies that $p_2 \cdot \theta, .., p_k \cdot \theta \subseteq d_i$, and by Lemma 94, 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 93 guarantees that for every atom in

$p_2, \ldots, p_k$ there is a variable not changed by θ that appears in a different location in p_1. Hence, $p_1 \cdot \theta \notin d_{i+1}$, contradicting our assumption that $d_i \preceq d_{i+1}$.

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

Theorem 96 *Let p be a predicate symbol of arity $a \geq 1$. There exists a proper subsumption chain of length $a^{\Omega(l)}$ of function free clauses using at most a variables and l literals if $l \leq a/2$.*

Proof. Lemma 90 guarantees that the chain produced is of length at least $N(a, l-1) \geq \binom{a}{l} - 1 = a^{\Omega(l)}$ if $l \leq a/2$. ■

In the remainder of this section, we strengthen the result of Theorem 96 achieving the same exponential bound using predicates of arity at most 3.

Definition 43 Let d be any clause. Let $\mathit{Trans}(d)$ be the clause obtained by replacing each literal $p(t_1, \ldots, t_a)$ with a new set $\{p(y_i, y_{i+1}, t_i) \mid 1 \leq i \leq a\}$, where all $y_1, \ldots, y_{a+1}$ are new variables not appearing in d. The new variables $y_1, \ldots, y_{a+1}$ should be different for each atom in d.

Example 16 The clause $p(z, x_1, x_2, z), p(z, z, x_1, z)$ is transformed into the clause
$p(y_1, y_2, z)$, $p(y_2, y_3, x_1)$, $p(y_3, y_4, x_2)$, $p(y_4, y_5, z)$,
$p(y'_1, y'_2, z)$, $p(y'_2, y'_3, z)$, $p(y'_3, y'_4, x_1)$, $p(y'_4, y'_5, z)$.

Lemma 97 *Let d be a function free clause with predicate symbols of arity at most a, containing at most v variables and l literals. Then, $Trans(d)$ uses predicates of arity 3, has $l(a+1)+v$ variables and uses at most al literals.* ■

Lemma 98 *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 see 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 $p(y'^*_h, y'^*_{h+1}, *)$ but in this case $i - 1 + 1 \neq i'$. If $i = 1$, then either we

find variables

$$\{y^j_{i+h} \mapsto y'^{j'}_{i'+h}, y^j_{i+h+1} \not\mapsto y'^{j'}_{i'+h+1}\} \in \theta$$

for some h s.t. $i + h \leq arity(p)$ in which case we arrive to the same contradiction as in the previous case. Otherwise, there is no such pair and hence

$$p(y^j_{arity(p)}, y^j_{arity(p)+1}, *) \cdot \theta = p(y'^{j'}_{arity(p)+i'}, y'^{j'}_{arity(p)+1+i'}, *) \notin Trans(d_2)$$

because the variable $y'^{j'}_{arity(p)+1+i'}$ does not exist in $Trans(d_2)$.

Now, the fact that each $y^j_i \mapsto y'^{j'}_i$ 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 $d_1 \cdot \theta \subseteq d_2$ and $d_1 \preceq d_2$ as required. ■

Corollary 99 *If there is a predicate symbol of arity at least* 3*, then there exist proper subsumption chains of length* $v^{\Omega(\sqrt{v})}$ *of function free clauses using at most* v *variables and* $\frac{v}{2}$ *literals, where* $v \geq 9$*.*

Proof. Theorem 96 shows that there exists a chain of length $\sqrt{v}^{\Omega(\sqrt{v})} = v^{\Omega(\sqrt{v})}$ 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 98 guarantees that this is also a proper chain. Obviously, it is of the same length as the initial one, and by Lemma 97 it uses 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. ■

8.2 Learning from membership queries only

In this section we show how a result on some aspect of the structure of first order clauses can be exploited to prove a negative learnability result. In this case, we

show that there can be no polynomial algorithm that learns the class of monotone function free clauses from membership queries only.

We use a combinatorial notion, the *teaching dimension* (Angluin, 2001; Goldman and Kearns, 1995) that is a lower bound for the complexity of exact learning from membership queries only.

Definition 44 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.

We show that the teaching dimension of the class of monotone first order clauses is of exponential size, thus eliminating the possibility of existence of a polynomial learning algorithm that has access to membership queries only.

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. We consider all the possible minimal generalizations[1] of the clause $p(\langle a, .., a \rangle)$, where the constant a occurs k times. Among them we find the clauses

$$
\begin{aligned}
C_0 &= p(\langle x, .., x \rangle) \\
C_1 &= p(\langle a, x, .., x \rangle) \vee p(\langle x, a, x, .., x \rangle) \vee .. \vee p(\langle x, .., x, a \rangle) \\
C_2 &= p(\langle a, a, x, .., x \rangle) \vee p(\langle a, x, a, x.., x \rangle) \vee .. \vee p(\langle x, .., x, a, a \rangle) \\
&\vdots
\end{aligned}
$$

[1] That is, clauses C that are strict generalizations of $p(\langle a, .., a \rangle)$ for which no other clause C' is such that $p(\langle a, .., a \rangle) \succ C' \succ C$.

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

Clearly, $|C_i| = \binom{k}{i}$. In particular, $\left|C_{k/2}\right| = \binom{k}{k/2} > 2^{k/2} > k^{\sqrt{k}}$.

We next define the learning problem for which we find an exponential lower bound. The signature $\mathcal{S}$ consists of the function symbols $\{f/2, a/0, b/0\}$ and a single predicate symbol $\{p/1\}$. Fix l to be some integer. Let the (representation) concept class be

$$\mathcal{C} = \{\text{first order monotone } \mathcal{S}\text{-clauses with at most } l \text{ atoms}\}$$

Let 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 $f = p(\langle a,..,a\rangle)$ 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: $l \leq \frac{\binom{k}{k/2}}{2} = \frac{\left|C_{k/2}\right|}{2} < \left|C_{k/2}\right|$. 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. There are exactly $K = \binom{\binom{k}{k/2}}{l} > (\frac{k^{\sqrt{k}}}{l})^l > k^{(\sqrt{k}-1)l} = k^{\Omega(l\sqrt{k})}$ such subsets, 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 $p(\langle a, .., a\rangle) \not\preceq E$. Hence, for each $C^j_{k/2}$ the example E^j must be included in T. Hence, T must contain each $E^1, \ldots, E^K$ and the teaching dimension for this class is at least $K = k^{\Omega(l\sqrt{k})}$.

Theorem 100 *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 at least $t^{\Omega(l\sqrt[4]{t})}$.*

Proof. Just set $k = \sqrt{t}$ and notice that clauses have at most $l \leq \frac{\sqrt{t}}{2}$ atoms and each atom contains at most $2k$ terms, hence a clause contains at most $2kl \leq 2\sqrt{t}\frac{\sqrt{t}}{2} = t$ terms. ■

Therefore we can conclude:

Corollary 101 *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, there is no polynomial algorithm that learns $\mathcal{C}$ from membership queries only.* ■

8.3 On the number of pairings

Next, we give a exponential lower bound on the number of pairings between two arbitrary clauses. In Chapter 5 we prove an (asymptotically) matching upper bound. We use the following basic fact:

Fact 102 Let $v \in \mathbb{N}$. Let π and π' be two distinct permutations of $\{0,..,v-1\}$. Then, there exists an index $l \in \{0,..,v-2\}$ such that for no other index $l' \in \{0,..,v-2\}$ it holds that $\pi(l) = \pi(l')$ and $\pi(l+1) = \pi(l'+1)$. In other words, when writing the permutations π, π' as an array of numbers $(\pi(0),..,\pi(v-1))$ and $(\pi'(0),..,\pi'(v-1))$, there must exist two consecutive terms $\pi(l), \pi(l+1)$ in $(\pi(0),..,\pi(v-1))$ that can not be found one after the other in $(\pi'(0),..,\pi'(v-1))$. ■

8.3.1 General clauses

In this section we show that general first order Horn clauses can have an exponential number of pairings.

Fix $v \in \mathcal{N}$ 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 $x_0,..,x_{v-1}$ and $y_0,..,y_{v-1}$ be variables. We define

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

and

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

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 $\{0,..,v-1\}$: each variable x_i in C_1 is matched to $y_{\pi(i)}$ in C_2. We implicitly assume that 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

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

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 indeed 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 $lgg_{\pi'}(C_1, C_2)$ — see Fact 102. Since the terms $t_{*,*}$ are ground, subsumption is not possible.

There are $v!$ distinct permutations of $\{0, \ldots, v-1\}$ so we conclude that there are $v!$ different pairings of C_1 and C_2. Hence:

Theorem 103 *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 $\Omega(v!)$.* ■

8.3.2 Function free clauses

Can we do the same trick without using function symbols? Our first attempt is to try to mimic 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

that the terms $t_{i,j}$ did. 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

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

and

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

Notice that $|C_1| = v^2(v-1)$ and $|C_2| = v^2$, they use a single predicate symbol of arity $2\log v + 2$, and both clauses use exactly $v+2$ variables.

Again, 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

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

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,

$$lgg_{\pi\cup\{z_0-z_0,z_1-z_1\}}(C_1,C_2) = \bigvee_{0\leq l<v-1} p(binary(\pi(l)),binary(\pi(l+1)),X_{\pi(l)},X_{\pi(l+1)}),$$

where we abuse notation and use $binary(n)$ to denote the tuple $z_{n_1},..,z_{n_{\log v}}$ encoding the integer n in its binary representation using z_0,z_1. For example, assuming

$v = 8$, $binary(6) = z_1, z_1, z_0$.

Example 17 Let $v = 4$ and let $\pi = (3201)$. Hence, in this example we use a predicate symbol $p/6$. 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

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

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

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

Clause C_2 is

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

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

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

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 indeed subsumption inequivalent, i.e., if for any $\pi \neq \pi'$

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

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 Fact 102 guarantees that 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 $p(z_0, \ldots, z_0, *, *)$ or $p(z_1, \ldots, z_1, *, *)$. 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, $\overline{binary(n)} = binary(v - 1 - n)$. Thus:

$$\begin{aligned}
& lgg_{\pi' \cup \{z_0 - z_0, z_1 - z_1\}}(C_1, C_2) \\
&= \bigvee_{0 \leq l < v-1} p(binary(\pi'(l)), binary(\pi'(l+1)), X_{\pi'(l)}, X_{\pi'(l+1)}) \\
&= \bigvee_{0 \leq l < v-1} p(\overline{binary(\pi(l))}, \overline{binary(\pi(l+1))}, X_{\pi'(l)}, X_{\pi'(l+1)}) \\
&= \bigvee_{0 \leq l < v-1} p(binary(v - 1 - \pi(l)), binary(v - 1 - \pi(l+1)), X_{\pi'(l)}, X_{\pi'(l+1)}) \\
&= \bigvee_{0 \leq l < v-1} p(binary(\overline{\pi}(l)), binary(\overline{\pi}(l+1)), X_{\overline{\pi}(l)}, X_{\overline{\pi}(l+1)}),
\end{aligned}$$

where $\overline{\pi}(l) = v - 1 - \pi(l)$, for all $1 \leq l < v$. 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 $\{z_0 \mapsto z_1, z_1 \mapsto z_0\} \cup \{X_l \mapsto X_{v-1-l} \mid 0 \leq l < v\}$.

There are $v!$ distinct permutations of $\{0, \ldots, v-1\}$ so we conclude that there are $\frac{v!}{2}$ different pairings of C_1 and C_2. Hence:

Theorem 104 *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 $\Omega(v!)$.* ■

8.3.3 Function free clauses with fixed arity

We strengthen the result in the previous section by finding a similar construction in which the arity is a fixed constant not dependent on v. We use Lemma 98, which gives us precisely a way to convert clauses with variable arity into fixed arity while preserving their subsumption properties.

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:

$$lgg_{M_\pi}(\mathit{Trans}(C_1), \mathit{Trans}(C_2)) \quad \approx \quad \mathit{Trans}(lgg_{\pi \cup \{z_0 - z_0, z_1 - z_1\}}(C_1, C_2)), \qquad (8.5)$$

where $\approx$ stands for the "variable renaming" relation.

In the previous section we established that there are $\frac{v!}{2}$ distinct pairings between C_1, C_2. Lemma 98 guarantees that the transformation on clauses $\mathit{Trans}(\cdot)$ preserves subsumption, hence there must be also $\frac{v!}{2}$ distinct clauses corresponding to the right hand side of Equation 8.5. Equation 8.5 therefore establishes that there are also $\frac{v!}{2}$ different pairings between $\mathit{Trans}(C_1)$ and $\mathit{Trans}(C_2)$. Moreover, the clauses $\mathit{Trans}(C_1)$ and $\mathit{Trans}(C_2)$ 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:

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

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

where

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

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

Intuitively, the clause $P_{l,i,j,x_l,x_{l+1}}$ uses the additional variables $\{u_k^{l,i,j}\}_{1\leq k\leq 2\log v+3}$ 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 $\{w_k^{i,j}\}_{1\leq k\leq 2\log v+3}$ 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 18 Following Example 17, 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,

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

$$\vee \quad p(u_5^{1,2,3}, u_6^{1,2,3}, x_1) \vee p(u_6^{1,2,3}, u_7^{1,2,3}, x_2)$$

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

Then $Trans(C_1) =$

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

$(u_1^{0,0,1},u_2^{0,0,1},z_0)$ $(u_2^{0,0,1},u_3^{0,0,1},z_0)$ $(u_3^{0,0,1},u_4^{0,0,1},z_0)$ $(u_4^{0,0,1},u_5^{0,0,1},z_1)$ $(u_5^{0,0,1},u_6^{0,0,1},x_0)$ $(u_6^{0,0,1},u_7^{0,0,1},x_1)$

$\vdots$

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

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

$(u_1^{1,0,0},u_2^{1,0,0},z_0)$ $(u_2^{1,0,0},u_3^{1,0,0},z_0)$ $(u_3^{1,0,0},u_4^{1,0,0},z_0)$ $(u_4^{1,0,0},u_5^{1,0,0},z_0)$ $(u_5^{1,0,0},u_6^{1,0,0},x_1)$ $(u_6^{1,0,0},u_7^{1,0,0},x_2)$

$(u_1^{1,0,1},u_2^{1,0,1},z_0)$ $(u_2^{1,0,1},u_3^{1,0,1},z_0)$ $(u_3^{1,0,1},u_4^{1,0,1},z_0)$ $(u_4^{1,0,1},u_5^{1,0,1},z_1)$ $(u_5^{1,0,1},u_6^{1,0,1},x_1)$ $(u_6^{1,0,1},u_7^{1,0,1},x_2)$

$\vdots$

$(u_1^{1,2,0},u_2^{1,2,0},z_1)$ $(u_2^{1,2,0},u_3^{1,2,0},z_0)$ $(u_3^{1,2,0},u_4^{1,2,0},z_0)$ $(u_4^{1,2,0},u_5^{1,2,0},z_0)$ $(u_5^{1,2,0},u_6^{1,2,0},x_1)$ $(u_6^{1,3,3},u_7^{1,2,0},x_2)$

$\vdots$

$(u_1^{1,3,3},u_2^{1,3,3},z_1)$ $(u_2^{1,3,3},u_3^{1,3,3},z_1)$ $(u_3^{1,3,3},u_4^{1,3,3},z_1)$ $(u_4^{1,3,3},u_5^{1,3,3},z_1)$ $(u_5^{1,3,3},u_6^{1,3,3},x_1)$ $(u_6^{1,3,3},u_7^{1,3,3},x_2)$

$(u_1^{2,0,0},u_2^{2,0,0},z_0)$ $(u_2^{2,0,0},u_3^{2,0,0},z_0)$ $(u_3^{2,0,0},u_4^{2,0,0},z_0)$ $(u_4^{2,0,0},u_5^{2,0,0},z_0)$ $(u_5^{2,0,0},u_6^{2,0,0},x_2)$ $(u_6^{2,0,0},u_7^{2,0,0},x_3)$

$(u_1^{2,0,1},u_2^{2,0,1},z_0)$ $(u_2^{2,0,1},u_3^{2,0,1},z_0)$ $(u_3^{2,0,1},u_4^{2,0,1},z_0)$ $(u_4^{2,0,1},u_5^{2,0,1},z_1)$ $(u_5^{2,0,1},u_6^{2,0,1},x_2)$ $(u_6^{2,0,1},u_7^{2,0,1},x_3)$

$$\vdots$$

$$(u_1^{2,3,3},u_2^{2,3,3},z_1)\ (u_2^{2,3,3},u_3^{2,3,3},z_1)\ (u_3^{2,3,3},u_4^{2,3,3},z_1)\ (u_4^{2,3,3},u_5^{2,3,3},z_1)\ (u_5^{2,3,3},u_6^{2,3,3},x_2)\ (u_6^{2,3,3},u_7^{2,3,3},x_3)$$

$Trans(C_2) =$

$$(w_1^{0,0},w_2^{0,0},z_0)\ (w_2^{0,0},w_3^{0,0},z_0)\ (w_3^{0,0},w_4^{0,0},z_0)\ (w_4^{0,0},w_5^{0,0},z_0)\ (w_5^{0,0},w_6^{0,0},y_0)\ (w_6^{0,0},w_7^{0,0},y_0)$$

$$\underline{(w_1^{0,1},w_2^{0,1},z_0)\ (w_2^{0,1},w_3^{0,1},z_0)\ (w_3^{0,1},w_4^{0,1},z_0)\ (w_4^{0,1},w_5^{0,1},z_1)\ (w_5^{0,1},w_6^{0,1},y_0)\ (w_6^{0,1},w_7^{0,1},y_1)}$$

$$(w_1^{0,2},w_2^{0,2},z_0)\ (w_2^{0,2},w_3^{0,2},z_0)\ (w_3^{0,2},w_4^{0,2},z_1)\ (w_4^{0,2},w_5^{0,2},z_0)\ (w_5^{0,2},w_6^{0,2},y_0)\ (w_6^{0,2},w_7^{0,2},y_2)$$

$$(w_1^{0,3},w_2^{0,3},z_0)\ (w_2^{0,3},w_3^{0,3},z_0)\ (w_3^{0,3},w_4^{0,3},z_1)\ (w_4^{0,3},w_5^{0,3},z_1)\ (w_5^{0,3},w_6^{0,3},y_0)\ (w_6^{0,3},w_7^{0,3},y_3)$$

$$(w_1^{1,0},w_2^{1,0},z_0)\ (w_2^{1,0},w_3^{1,0},z_1)\ (w_3^{1,0},w_4^{1,0},z_0)\ (w_4^{1,0},w_5^{1,0},z_0)\ (w_5^{1,0},w_6^{1,0},y_1)\ (w_6^{1,0},w_7^{1,0},y_0)$$

$$(w_1^{1,1},w_2^{1,1},z_0)\ (w_2^{1,1},w_3^{1,1},z_1)\ (w_3^{1,1},w_4^{1,1},z_0)\ (w_4^{1,1},w_5^{1,1},z_1)\ (w_5^{1,1},w_6^{1,1},y_1)\ (w_6^{1,1},w_7^{1,1},y_1)$$

$$(w_1^{1,2},w_2^{1,2},z_0)\ (w_2^{1,2},w_3^{1,2},z_1)\ (w_3^{1,2},w_4^{1,2},z_1)\ (w_4^{1,2},w_5^{1,2},z_0)\ (w_5^{1,2},w_6^{1,2},y_1)\ (w_6^{1,2},w_7^{1,2},y_2)$$

$$(w_1^{1,3},w_2^{1,3},z_0)\ (w_2^{1,3},w_3^{1,3},z_1)\ (w_3^{1,3},w_4^{1,3},z_1)\ (w_4^{1,3},w_5^{1,3},z_1)\ (w_5^{1,3},w_6^{1,3},y_1)\ (w_6^{1,3},w_7^{1,3},y_3)$$

$$\underline{(w_1^{2,0},w_2^{2,0},z_1)\ (w_2^{2,0},w_3^{2,0},z_0)\ (w_3^{2,0},w_4^{2,0},z_0)\ (w_4^{2,0},w_5^{2,0},z_0)\ (w_5^{2,0},w_6^{2,0},y_2)\ (w_6^{2,0},w_7^{2,0},y_0)}$$

$$(w_1^{2,1},w_2^{2,1},z_1)\ (w_2^{2,1},w_3^{2,1},z_0)\ (w_3^{2,1},w_4^{2,1},z_0)\ (w_4^{2,1},w_5^{2,1},z_1)\ (w_5^{2,1},w_6^{2,1},y_2)\ (w_6^{2,1},w_7^{2,1},y_1)$$

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

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

$$(w_1^{3,0},w_2^{3,0},z_1)\ (w_2^{3,0},w_3^{3,0},z_1)\ (w_3^{3,0},w_4^{3,0},z_0)\ (w_4^{3,0},w_5^{3,0},z_0)\ (w_5^{3,0},w_6^{3,0},y_3)\ (w_6^{3,0},w_7^{3,0},y_0)$$

$$(w_1^{3,1},w_2^{3,1},z_1)\ (w_2^{3,1},w_3^{3,1},z_1)\ (w_3^{3,1},w_4^{3,1},z_0)\ (w_4^{3,1},w_5^{3,1},z_1)\ (w_5^{3,1},w_6^{3,1},y_3)\ (w_6^{3,1},w_7^{3,1},y_1)$$

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

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

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

$Trans(C_2)$ as follows:

$$\{x_i - y_{\pi(i)} \Rightarrow X_{\pi(i)}\}_{0\leq i<v} \cup \{z_0 - z_0, z_1 - z_1\} \cup \tag{8.6}$$

$$\{u_k^{l,\pi(l),\pi(l+1)} - w_k^{\pi(l),\pi(l+1)} \Rightarrow W_k^l\}_{1\leq k\leq 2\log v+3 \text{ and } 0\leq l<v-1} \cup \tag{8.7}$$

$$\{u_k^{0,i,j} - w_{2\log v+4-k}^{i,j}\}_{1\leq k\leq 2\log v+3 \text{ and } (i,j)\in[v]^2\setminus\{(\pi(l),\pi(l+1))\,|\,0\leq l<v-1\}} \tag{8.8}$$

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 (8.7) and (8.8) determine the matchings between auxiliary variables (those coming from the transformation *Trans*); part (8.6) matches original variables. As we see next, (8.7) and (8.8) are designed so that atoms in $lgg_{\pi\cup\{z_0-z_0,z_1-z_1\}}(C_1, C_2)$ survive[2] — this is done by (8.7) — and everything else disappears (8.8).

We carefully study $lgg_{M_\pi}(Trans(C_1), Trans(C_2))$. We observe that M_π matches auxiliary variables $u_*^{*,i,j} - w_*^{i,j}$. This has the effect that in the resulting pairing, only atoms coming from clauses $P_{*,i,j,*,*} \in Trans(C_1)$ and $P_{i,j,**} \in Trans(C_2)$ survive. Hence, it suffices to study the effect of the matching on clauses $P_{*,i,j,*,*} \times P_{i,j,**}$ only. In the case that $i = \pi(l)$ and $j = \pi(l+1)$ for some $l \in \{0, \ldots, v-2\}$, we observe that the auxiliary variables are matched following their order in the chain $\{u_k^{l,\pi(l),\pi(l+1)} - w_k^{\pi(l),\pi(l+1)} \Rightarrow W_k^l\}_{1\leq k\leq 2\log v+3}$ (8.7), and hence clauses $P_{l,\pi(l),\pi(l+1),x_l,x_{l+1}} \in C_1$ and $P_{\pi(l),\pi(l+1),y_{\pi(l)},y_{\pi(l+1)}} \in C_2$ survive 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 $W_1^l, \ldots, W_{2\log v+3}^l$. To see that atoms in the product $P_{*,i,j,*,*} \times P_{i,j,**}$ do not survive when $(i,j) \notin [v]^2 \setminus \{(\pi(l), \pi(l+1)) \mid 0 \leq l < v-1\}$, it is sufficient to observe that the auxiliary vari-

[2]By a "surviving literal" we mean a literal that is the product of literals in the respective clauses included in the pairing because their arguments are matched according to the 1-1 matching inducing the pairing.

ables are matched in reversed order $\{u_k^{0,i,j} - w_{2\log v+4-k}^{i,j}\}_k$ (8.8), so that in order to survive 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:

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

Example 19 Following Example 18, the matching $M_{(3201)}$ is as follows. Corresponding to 8.6:

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

Corresponding to 8.7:

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

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

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

Corresponding to 8.8:

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

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

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

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

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

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

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

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

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 $\mathit{Trans}(C_1)$

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

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

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

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

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

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

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

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

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

the reader 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 $\{Y_k^1 \leftrightarrow W_k^0, Y_k^2 \leftrightarrow W_k^1, Y_k^3 \leftrightarrow W_k^2 \mid 1 \leq k \leq 7\}$.

Theorem 105 *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 $\Omega(v!)$.* ■

Corollary 106 *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 $\Omega(2^{v/4})$ for sufficiently large v.* ■

Chapter 9

Conclusions and Future Work

In this thesis we have studied the complexity of learning first order and propositional classes in the model of exact learning from queries. An upper bound for the first order problem has been obtained by constructing the algorithm Learn-Closed-Horn in Chapter 5 that learns an interesting subclass of first order Horn expressions. Its complexity is exponential in two parameters: a (the maximal arity of the predicates used) and v (the bound on the number of variables permitted in any clause). The natural question after presenting this algorithm is whether it is optimal in the sense that this exponential dependence is necessary or whether better (polynomial) learning algorithms exist. Chapter 6 tries to answer this question by characterizing the VC Dimension of the class of first order Horn expressions which is known to give a lower bound for the complexity of learning in our model. However the VC Dimension is $\tilde{\Theta}(cl + ct)$ so that it gives a lower bound of $\tilde{\Omega}(cl + ct)$. Hence, the VC Dimension cannot settle our question.

While studying the VC Dimension of first order Horn expressions, we realized that there was a disparity between how the complexity of the expressions was measured in the learning algorithms found in the literature (that used first order syntactic properties such as the number of variables or number of clauses, etc.) and how the formal definitions were presented (where the notion of size is used without explaining how to measure it). At that point it was not clear what the best way to

measure this complexity was. Chapter 4 clarifies this question, with the conclusion that two fundamentally different ways of measuring the complexity exist: what we call *TreeSize* and what we call *DAGSize*. *TreeSize* is considered more standard and it is closely related to the number of symbols needed to write a first order expression in its usual form; *DAGSize* encodes the term in a smarter way by allowing shared terms to be represented just once. In Chapter 4 we show that three parameters, the number of clauses (c), number of terms per clause (t), and number of literals per clause (l), capture the notion of *DAGSize*. However, in the case of *TreeSize*, none of the parameters usually considered can capture it. From this last observation we conclude that none of the existing results on learnability of first order expressions are valid if one considers *TreeSize* as the way of measuring the complexity of first order expressions. Surprisingly, this fact had never been noticed before.

Returning to the question of whether our learning algorithm of Chapter 5 is optimal, we have seen that the VC Dimension does not give a complete answer. Hence, a more powerful tool, the certificate size, needs to be considered. As a first step towards characterizing the certificate size of first order Horn expressions, we compute in Chapter 7 the certificate size of various propositional classes. In some cases we are even able to give exact characterizations: for example, Theorem 77 and Theorem 65 show that the certificate size of unate DNF formulas is exactly $m + 1 + \binom{m+1}{2}$ if $m < n$. However, if $m \geq n$, the result obtained is weaker: the upper bound is $O(mn)$ (Theorem 63) but the lower bound of $\Omega(nm)$ applies to the *strong* certificate size only, which is a weaker version of the certificate size. To obtain a complete characterization of the certificate size in the case $m \geq n$, we need to obtain a strong version of our Theorem 78 (similar to the stronger Theorem 77 version of Theorem 76). When quantifying the certificate size of Horn CNF expressions where $m \geq n$, we have not only that the lower bound of $\Omega(mn)$ applies to the strong certificate size only, but there exists a gap to the upper bound which is $O(m^2)$. Here, the question of whether we can prove a higher lower bound or else if we can create certificates of smaller size to match the lower bound is still

open. Our final result involving certificates answers an open question by Feigelson (1998): a slight generalization of Horn CNF, renamable Horn CNF, does not have polynomial certificates and is therefore not learnable in the model of exact learning from membership and equivalence queries.

Clearly, the question of whether our algorithm Learn-Closed-Horn is optimal remains open. It is possible that our learning algorithm is optimal but our analysis is not tight. Towards this we compute in Section 8.3 lower bounds for the number of pairings between two clauses, which is the main reason for the algorithm's exponential dependence on v. Our construction shows that there are classes for which the number of pairings is indeed exponential in v, thus showing that the complexity analysis is tight.

A way of answering the question of the algorithm's optimality is by computing the certificate size of first order Horn expressions. This is important not only because of its applications to learnability, but also from the point of view of a logician as it would provide great insight into the structure of the very important class of first order Horn expressions. A first attempt of generalizing the construction in Theorem 73 of Chapter 7 led us to the study of the length of proper chains of first order clauses w.r.t. the subsumption relation which appears in Chapter 8. Our initial generalization attempt failed due to technical subtleties, however, we could still prove a weaker result: no polynomial learning algorithm can exist if just membership queries are allowed (Section 8.2).

We conclude by mentioning broader challenges for the future. The first is concerned with establishing the theoretical boundaries of what is considered efficiently learnable. For example, in our particular learning setting and problem, we have two possible scenarios. If it turns out that our algorithm is optimal, then no polynomial algorithm can exist for our class. Hence, we should identify which restrictions of the class of closed Horn expressions are learnable with polynomial complexity. On the other hand, if our algorithm is not optimal and polynomial learning is possible, then we should identify more general classes for which efficient learning is still possible.

The second general challenge is concerned with how to apply query-learning algorithms in practice. In the introduction we have mentioned how some existing systems do this: by simulating the queries using a database of examples, by performing actual experiments or simulations, or by seeking human help. Here the challenge is to identify new domains where this is possible and beneficial.

Bibliography

Angluin, D. 1987a. Learning k-term DNF formulas using queries and counterexamples. Technical Report YALEU/DCS/RR-559, Department of Computer Science, Yale University, August.

Angluin, D. 1987b. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, November.

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

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

Angluin, Dana. 2001. 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. Springer.

Angluin, Dana, Lisa Hellerstein, and Marek Karpinski. 1993. Learning read-once formulas with queries. *Journal of the ACM*, 40(1):185–210, January.

Arias, M. and R. Khardon. 2000. Learning Inequated Range Restricted Horn Expressions. Technical Report EDI-INF-RR-0011, Division of Informatics, University of Edinburgh, March.

Arias, M. and R. Khardon. 2002. Learning closed horn expressions. *Information and Computation*, 178:214–240.

Arimura, Hiroki. 1997. Learning acyclic first-order Horn sentences from entailment. In *Proceedings of the International Conference on Algorithmic Learning Theory*, Sendai, Japan. Springer-Verlag. LNAI 1316.

Balcázar, José L., Jorge Castro, and David Guijarro. 1999. The consistency dimension and distribution-dependent learning from queries. In *Proceedings of the International Conference on Algorithmic Learning Theory*, Tokyo, Japan, December 6-8. Springer. LNAI 1702.

Blumer, Anselm, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. 1989. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, October.

Bryant, C. and S. Muggleton. 2000. Closed loop machine learning. Technical Report YCS 330, University of York, Department of Computer Science, York, U.K.

Bshouty, Nader H. 1995. Simple learning algorithms using divide and conquer. In *Proceedings of the Conference on Computational Learning Theory*.

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

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

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

De Raedt, L. and M. Bruynooghe. 1992. An overview of the interactive concept-learner and theory revisor CLINT. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, pages 163–192.

De Raedt, L. and W. Van Laer. 1995. Inductive constraint logic. In *Proceedings of the 6th Conference on Algorithmic Learning Theory*, volume 997. Springer-Verlag.

Ehrenfeucht, Andrzej, David Haussler, Michael Kearns, and Leslie Valiant. 1989. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82(3):247–251, September.

Feigelson, Aaron. 1998. *On Boolean Functions and their Orientations: Learning, monotone dimension and certificates*. Ph.D. thesis, Northwestern University, Evanston, IL, USA, June.

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

Goldman, Sally A. and Michael Kearns. 1995. On the complexity of teaching. *Journal of Computer and System Sciences*, 50:20–31.

Gottlob, Georg and Christos Papadimitriou. 2003. On the complexity of single-rule datalog queries. *Inf. Comput.*, 183(1):104–122.

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

Hegedus, T. 1995. 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. ACM Press.

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

Hellerstein, Lisa and Vijay Raghavan. 2002. Exact learning of DNF formulas using DNF hypotheses. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC-02)*, pages 465–473, New York, May 19–21. ACM Press.

Horn, A. 1956. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16:14–21.

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

Khardon, R. 1999b. Learning range restricted Horn expressions. In *Proceedings of the Fourth European Conference on Computational Learning Theory*, pages 111–125, Nordkirchen, Germany. Springer-verlag. LNAI 1572.

Khardon, R. and D. Roth. 1999. Learning to reason with a restricted view. *Machine Learning*, 35(2):95–117.

Khardon, Roni. 2000. Learning horn expressions with LogAn-H. In *Proceedings of the International Conference on Machine Learning*, pages 471–478. Morgan Kaufmann.

Khardon, Roni and Dan Roth. 1996. Reasoning with models. *Artificial Intelligence*, 87(1–2):187–213.

Kietz, J-U. and M. Lübbe. 1994. An efficient subsumption algorithm for inductive logic programming. In S. Wrobel, editor, *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237, pages 97–106. Gesellschaft für Mathematik und Datenverarbeitung MBH.

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

Maass, Wolfgang and György Turán. 1992. Lower bound methods and separation results for online learning models. *Machine Learning*, 9:107–145, October.

Marcinkowski, J. and L. Pacholski. 1995. Undecidability of the horn clause implication problem. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pages 354–362, Pittsburgh, PA, USA.

McKinsey, J. C. C. 1943. The decision problem for some classes of sentences without quantifiers. *J. Symbolic Logic*, 8:61–76.

Mitchell, T. 1997. *Machine Learning*. McGraw-Hill.

Muggleton, S. 1995. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286.

Muggleton, S., C. Bryant, C. Page, and M. Sternberg. 1999. Combining active learning with inductive logic programming to close the loop in machine learning. In *Proceedings of the AISB'99 Symposium on AI and Scientic Creativity*.

Muggleton, S. and W. Buntine. 1988. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning*, pages 339–351. Morgan Kaufmann.

Muggleton, S. and C. Feng. 1992. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*. Academic Press, pages 281–298.

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

Papadimitriou, Christos H. and Mihalis Yannakakis. 1997. On the complexity of database queries (extended abstract). In *Proceedings of the 16th Annual ACM Symposium on Principles of Database Systems*, pages 12–19. ACM Press.

Pillaipakkamnatt, Krishnan and Vijay Raghavan. 1996. On the limits of proper learnability of subclasses of DNF formulas. *Machine Learning*, 25:237.

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

Plotkin, G. D. 1971. A further note on inductive generalization. *Machine Intelligence*, 6:101–124.

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

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

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

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

Reddy, C. and P. Tadepalli. 1999. Learning Horn definitions: Theory and an application to planning. *New Generation Computing*, 17:77–98.

Sammut, C. and R. Banerji. 1986. Learning concepts by asking questions. In R. Michalski, J. Carbonnel, , and T. Mitchell, editors, *Machine Learning: An Articial Intelligence Approach*. Morgan Kaufmann, pages 167–192.

Schmidt-Schauss, M. 1988. Implication of clauses is undecidable. *Theoretical Computer Science*, 59:287–296.

Semeraro, G., F. Esposito, D. Malerba, and N. Fanizzi. 1998. A logic framework for the incremental inductive synthesis of datalog theories. In *Proceedings of the International Conference on Logic Program Synthesis and Transformation (LOPSTR'97)*. Springer-Verlag. LNAI 1463.

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

Valiant, L. G. 1985. Learning disjunctions and conjunctions. In *Proceedings of the International Joint Conference in Artificial Intelligence*, pages 560–566. Morgan Kaufmann.

Valiant, Leslie G. 1984. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November.

Vardi, M. Y. 1982. The complexity of relational query languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC-82)*, pages 137–146, New York. ACM Press.