On Learning Conjunctions of Horn$^{\supset}$ Clauses *

J. Gaintzarain, M. Hermo and M. Navarro

Dpto de L.S.I., Facultad de Informática,
P.O. Box 649, 20080-San Sebastián, SPAIN.

Abstract. $Horn^{\supset}$ is a logic programming language which extends usual $Horn$ clauses with intuitionistic implication. In this paper we study the learnability of the concepts that are represented by conjunctions of propositional $Horn^{\supset}$ clauses.

1 Introduction

The problem of learning efficiently an unknown boolean formula under some determined protocols has been widely studied. It is well known that, even restricted to propositional formulas, the problem is hard [3, 10] in the usual learning models. Among these, we find the *PAC model* introduced by Valiant [19] and the *prediction model* introduced by Pitt and Warmuth [18]. Both are *passive* models in the sense that the learning algorithm has no control over the labeled examples, however they can be transformed into *active* ones by adding the ability to make membership queries. Another usual model is the *exact query model* introduced by Angluin [1], where the learning algorithm can make equivalence and/or membership queries.

In the line of learning subclasses of boolean formulas, D. Angluin, M. Frazier and L. Pitt [2] presented a positive result: a polynomial-time algorithm that used equivalence and membership queries for exactly learning conjunctions of propositional $Horn$ clauses. In this field we can also find negative results showing that there are not efficient learning algorithms for a given class. Even there exist relative results which, without solving the problem, give evidence to the effect that some class is not learnable. This kind of relative result is based on the concept of prediction-preserving reduction [18, 3].

From the point of view of logic programming, different approaches for extending Horn clauses have been studied. Some of them consider to incorporate into the language a new implication symbol, $\supset$, with the aim of structuring logic programs in some blocks with local clauses [4, 7–9, 13–17]. These extensions can also be seen as a sort of inner modularity in logic programming [5].

We consider here a particular extension named $Horn^{\supset}$. This programming language has been formally studied in [9, 8, 4, 11, 17]. In [4] a natural extension of classical first order logic $\mathcal{FO}$ with the intuitionistic implication ($\supset$), named $\mathcal{FO}^{\supset}$, was presented as the underlying logic of the programming language $Horn^{\supset}$. It was also proved that the "good properties" that verify $Horn$ clauses (as a

* This work has been partially supported by CICYT-project TIC2001-2476-C03-03

programming language) with respect to its underlying logic $\mathcal{FO}$ are conserved by $Horn^{\supset}$ with respect to $\mathcal{FO}^{\supset}$. In concrete, general models can be restricted to Herbrand-like models, each program has a canonical model and the operational semantics is an effective subcalculus of a complete calculus for $\mathcal{FO}^{\supset}$ (see also [12, 11]).

Our aim is to study the learnability of propositional $Horn^{\supset}$ clauses since this class of formulas, extending propositional $Horn$ clauses, seems to be a good candidate for improving the learnability results obtained by D. Angluin, M. Frazier and L. Pitt [2].

This paper presents our first results on such study and it is organized as follows: In Section 2 the preliminary notions in the area of learning are given. In Section 3 the programming language $Horn^{\supset}$ (restricted to the propositional case) is presented. In Section 4 we show that any model-based translation from $Horn^{\supset}$ clauses to equivalent $Horn$ clauses obtains, in general, an exponential number of clauses. This result suggests us that finding a polynomial-time algorithm for learning $Horn^{\supset}$ language may be difficult. In section 5 we show that boolean formulas are prediction preserving reducible to $Horn^{\supset}$ clauses using membership queries. As a consequence, we obtain the main result of this paper: conjunctions of propositional $Horn^{\supset}$ clauses are not predictable even with membership queries under cryptographic assumptions. We conclude, in Section 6, by summarizing our results.

2 Learning preliminaries

Most of the terminology used in this section is borrowed from [3, 6]. Let S be a fixed domain and let S^* be the set of strings over S. $|w|$ denotes the length of the string w. Strings in S^* will represent both examples and concept names.

A *representation of concepts* (or *representation class*) $\mathcal{C}$ is any subset of $S^* \times S^*$. We interpret an element $\langle u, x\rangle$ of $S^* \times S^*$ as consisting of a *concept name* u and an *example* x. For instance, the representation of concepts $\mathcal{C}_{BF}$ contains pairs $\langle u, x\rangle$ where u is a boolean formula and x is any interpretation satisfying such formula. Define the *concept represented by* u as $K_{\mathcal{C}}(u) = \{x \mid \langle u, x\rangle \in \mathcal{C}\}$. The set of *concepts represented by* $\mathcal{C}$ is $\{K_{\mathcal{C}}(u) \mid u \in S^*\}$.

We use two models of learning, both of them fairly standard: Angluin's model of exact learning with queries [1] and the model of prediction with membership queries as defined by Angluin and Kharitonov [3].

2.1 The exact query learning model

Let $\mathcal{C}$ be a representation class. An *exact learning algorithm with queries* is an algorithm A that takes as input a bound s on the size of the target concept representation. It may make any number of queries or requests, the responses to which are determined by the unknown target concept c. The algorithm A must eventually halt with an output concept name v. The concept $K_{\mathcal{C}}(v)$ is interpreted as a A's guess of the target concept. The most common kinds of queries are

membership and *equivalence* queries. A *membership query* takes a string x in S^* as input and returns 1 if $x \in c$ and 0 otherwise. An *equivalence query* takes a concept name h as input and returns 1 if $K_{\mathcal{C}}(h) = c$ and a counterexample, in the symmetric difference of c and $K_{\mathcal{C}}(h)$, otherwise.

The algorithm A runs in polynomial time if its running time (counting one step for each query) is bounded by a polynomial in s and the length of the largest counterexample. We say that A *exactly learns* a representation of concepts $\mathcal{C}$, if and only if for all positive integer s, for all concept name u with $|u| \leq s$, when A runs with input s and a target concept $c = K_{\mathcal{C}}(u)$, A outputs a concept name v such that $c = K_{\mathcal{C}}(v)$

A representation of concepts $\mathcal{C}$ is *polynomially learnable* if there is a learning algorithm A that runs in polynomial time and exactly learns $\mathcal{C}$. In this paper, we always suppose that A uses membership and equivalence queries.

2.2 The prediction model with membership queries

A *prediction with membership algorithm*, or *pwm-algorithm* is a possible randomized algorithm A that takes as input a bound s as above, a bound n on the length of examples, and an accuracy bound ϵ. It may make three different kinds of queries or requests, the responses to which are determined by the unknown target concept c and an unknown probability distribution $\mathcal{D}$ on S^n, as follows:
– A *membership query* takes a string x in S^* as input and returns 1 if $x \in c$ and 0 otherwise.
– A *request for a random classified example* takes no input and return a pair $\langle x, b \rangle$, where x is a string chosen independently according to $\mathcal{D}$ and $b = 1$ if $x \in c$ and $b = 0$ otherwise.
– A *request for an element to predict* takes no input and returns a string *sol* chosen independently according to $\mathcal{D}$.

The algorithm A may make any number of membership queries or requests for random classified examples. However, A must eventually make one and only one request for an element to predict and eventually halt with an output 1 or 0 without making any further query or request. The output is interpreted as A's guess of how the target concept classifies the element *sol*. We say that A runs in polynomial time if its running time (counting one step for each query or request) is bounded by a polynomial in s, n, and $\frac{1}{\epsilon}$.

We say that A *predicts* a representation of concepts $\mathcal{C}$ if and only if for all positive integers s and n, for all positive rational ϵ, for all concept name $u \in S^*$ with $|u| \leq s$, when A runs with inputs s, n, and ϵ, and a target concept $c = K_{\mathcal{C}}(u)$ and $\mathcal{D}$, A asks queries or requests and the probability that the output of A is not equal to the correct classification of *sol* by c is at most ϵ.

A representation of concepts $\mathcal{C}$ is *polynomially predictable with membership queries* if and only if there is an algorithm A that runs in polynomial time and predicts $\mathcal{C}$.

Lemma 1. *[3] If a representation of concepts is polynomially learnable, then it is polynomially predictable with membership queries.*

2.3 Reducibility among prediction problems

To compare the difficulty of learning problems in the prediction model we use *pwm-reducibility* as defined in [3]. It is denoted by $\leq_{pwm}$.

Definition 1. *Let $\mathcal{C}$ and $\mathcal{C}'$ be representations of concepts. Let $\top$ and $\bot$ be two different symbols not occurring in S. Then $\mathcal{C}$ is* pwm-reducible *to $\mathcal{C}'$, if and only if there exist three mappings g, f and h with the following properties:*

1. There is a nondecreasing polynomial q such that for all natural numbers s and n and for every $u \in S^*$ with $|u| \leq s$, $g(s, n, u)$ is a string u' of length at most $q(s, n, |u|)$.

2. For all natural numbers s and n, for every $u \in S^*$ with $|u| \leq s$, and for every $x \in S^*$ with $|x| \leq n$, $f(s, n, x)$ is a string x' and $x \in K_{\mathcal{C}}(u)$ if and only if $x' \in K_{\mathcal{C}'}(g(s, n, u))$.

3. For all natural numbers s and n, for every $u \in S^*$ with $|u| \leq s$, and for every $x' \in S^*$, $h(s, n, x')$ is either $\top$, $\bot$ or a string x. If $h(s, n, x') = \top$ then $x' \in K_{\mathcal{C}'}(g(s, n, u))$; if $h(s, n, x') = \bot$ then $x' \notin K_{\mathcal{C}'}(g(s, n, u))$; and otherwise $x' \in K_{\mathcal{C}'}(g(s, n, u))$ if and only if $x \in K_{\mathcal{C}}(u)$. Moreover, h is computable in time bounded by a polynomial in s, n and $|x'|$.

In the property (2), and independently, in the property (3), the expression "$x \in K_{\mathcal{C}}(u)$" can be replaced with "$x \notin K_{\mathcal{C}}(u)$", as discussed in [3]. We denote these options by properties (2)' and (3)' respectively.

The only properties of this reducibility that are needed in this paper were established in [3]:

Lemma 2. *The pwm-reduction is transitive, i.e., let $\mathcal{C}$, $\mathcal{C}'$, and $\mathcal{C}''$ be representations of concepts, if $\mathcal{C} \leq_{pwm} \mathcal{C}' \leq_{pwm} \mathcal{C}''$ then $\mathcal{C} \leq_{pwm} \mathcal{C}''$.*

Lemma 3. *Let $\mathcal{C}$ and $\mathcal{C}'$ be representations of concepts. If $\mathcal{C} \leq_{pwm} \mathcal{C}'$ and $\mathcal{C}'$ is polynomially predictable with membership queries, then $\mathcal{C}$ is also polynomially predictable with membership queries.*

3 The programming language $Horn^{\supset}$

In this section we introduce the programming language $Horn^{\supset}$ by showing its syntax and its model semantics. Although the language is in general a first order language (see [9, 4]), in this paper we shall restrict our presentation only to this language in the propositional setting.

3.1 The syntax

The syntax is an extension of the (propositional) $Horn$ clause language by adding the intuitionistic implication $\supset$ in goals and clause bodies. Let Σ be a set of propositional variables (or signature). The Σ-clauses, named D, and the Σ-goals, named G, are recursively defined as follows (where v stands for any propositional variable in Σ):

$$G ::= v \mid G_1 \wedge G_2 \mid D \supset G \qquad\qquad D ::= v \mid G \to v \mid D_1 \wedge D_2$$

A $Horn^{\supset}$ Σ-program is a finite set (or conjunction) of Σ-clauses. The main difference with respect to $Horn$ clauses is the use of a "local" clause set D in goals of the kind $D \supset G$ (and therefore also in clause bodies).

Example 1. The following set with three clauses is a $Horn^{\supset}$ program over signature $\Sigma = \{a, b, c, d\}$

$$\{((b \to c) \supset c) \to a,\ b,\ ((a \wedge (b \to c)) \supset (((b \to c) \wedge (a \to d)) \supset a)) \to d\}$$

The second program clause is simply b. The first and the third program clauses are of the form $G \to v$. In the first clause, the goal G is $(b \to c) \supset c$. That is, it contains a local set with one program clause. In the third clause, the goal G is of the form $D_1 \supset (D_2 \supset G_3)$, where $D_1 = a \wedge (b \to c)$, $D_2 = (b \to c) \wedge (a \to d)$, and $G_3 = a$. D_1 and D_2 are local sets with two clauses and they can also be written $D_1 = \{a, (b \to c)\}$ and $D_2 = \{(b \to c), (a \to d)\}$ respectively.

3.2 The model semantics

Model semantics for the programming language $Horn^{\supset}$ is based on some Kripke structures which, in the propositional setting, can be defined as follows.

Definition 2. *Given a signature Σ, the model semantics for the propositional $Horn^{\supset}$ is given by the set of Kripke Σ-structures $\underline{\mathsf{Mod}}(\Sigma) = \{K_I \mid I \subseteq \Sigma\}$ where K_I denotes the partially ordered set of worlds $\{J \subseteq \Sigma \mid I \subseteq J\}$.*

Well-formed Σ-formulas are built, from propositional variables in Σ, using classical connectives ($\neg$, $\wedge$, $\vee$ and $\to$) and the intuitionistic implication ($\supset$). The satisfaction relation $\models_\Sigma$ between a Σ-structure K_I and a Σ-formula φ requires the formula φ to be forced in the minimal world I in K_I. The forcing relation is defined between (the associated interpretation to) a world and a formula.

Definition 3. *Let $K_I \in \underline{\mathsf{Mod}}(\Sigma)$ and φ a Σ-formula. We say that*

(a) *$K_I \models_\Sigma \varphi$ (K_I satisfies φ) iff $I \Vdash_\Sigma \varphi$ (φ is forced in I)*
(b) *The binary* forcing relation *$\Vdash_\Sigma$ (or simply $\Vdash$ if there is no confusion about the signature) is inductively defined as follows:*
$I \nVdash False$
$I \Vdash v$ iff $v \in I$ for $v \in \Sigma$
$I \Vdash \neg\varphi$ iff $I \nVdash \varphi$
$I \Vdash \varphi \wedge \psi$ iff $I \Vdash \varphi$ and $I \Vdash \psi$
$I \Vdash \varphi \vee \psi$ iff $I \Vdash \varphi$ or $I \Vdash \psi$
$I \Vdash \varphi \to \psi$ iff if $I \Vdash \varphi$ then $I \Vdash \psi$
$I \Vdash \varphi \supset \psi$ iff for all $J \subseteq \Sigma$ such that $I \subseteq J$: if $J \Vdash \varphi$ then $J \Vdash \psi$

Example 2. Let φ be the formula (in this case a goal) $((a \wedge c) \to b) \supset (c \wedge b)$. $I \Vdash \varphi$ for $I = \{a, b, c\}$, $I = \{a, c\}$ and $I = \{b, c\}$. $I \nVdash \varphi$ for $I = \{a, b\}$, $I = \{a\}$, $I = \{b\}$, $I = \{c\}$ and $I = \emptyset$. Note, for instance, that $\{a, b\} \Vdash (a \wedge c) \to b$ and $\{a, b\} \nVdash (c \wedge b)$.

This forcing relation does not behave monotonically with respect to the world ordering for general formulas. For instance, $a \to b$ is forced in the world $I = \emptyset$ but it is not forced in $J = \{a\}$. We say that a formula is *persistent* whenever the forcing relation behaves monotonically for it.

Definition 4. *A Σ-formula φ is* persistent *when for any Σ-interpretation I, if $I \Vdash \varphi$ then $J \Vdash \varphi$ for any Σ-interpretation J such that $I \subseteq J$.*

From the following proposition we obtain, in particular, that any Σ-goal G is a persistent formula.

Proposition 1. *[4] Any $v \in \Sigma$ is persistent. Any formula $\varphi \supset \psi$ is persistent. If φ and ψ are persistent then $\varphi \vee \psi$ and $\varphi \wedge \psi$ are persistent.*

Definition 5. *Two formulas φ and ψ are* semantically equivalent *if both have the same meaning in each structure in $\underline{\mathsf{Mod}}(\Sigma)$. In other words, if both are forced in the same Σ-interpretations.*

4 Trying to learn $Horn^{\supset}$ programs with $Horn$ programs

A way to learn a $Horn^{\supset}$ program P might consist on using the well-known algorithm of Angluin, Frazier and Pitt [2] for learning a conjunction of $Horn$ clauses equivalent to P. In this section, we show that this simple idea does not work if we are looking for a learning algorithm that runs in polynomial time. We prove that any translation from a $Horn^{\supset}$ Σ-program P to an equivalent $Horn$ Σ-program $\widehat{P}$ yields a number of clauses that is exponential in the size of P.

Definition 6. *For each $Horn^{\supset}$ Σ-clause D, let $Models(D)$ be the set $\{I \subseteq \Sigma \mid I \Vdash D\}$. Let $Min(D)$ be the set $\{I \subseteq \Sigma \mid I \not\Vdash D$ but $J \Vdash D$, for all $J \subset I\}$. That is, $Min(D)$ contains the "minimal" interpretations not forcing D.*

In the sequel, we intentionally consider I as a set or as a conjunction, as convenient.

Definition 7. *For each $Horn^{\supset}$ Σ-clause D, the $Horn$ Σ-program $\widehat{D}$ is defined as follows:*

For $D = v$, $\quad \widehat{D} = \{v\}$

For $D = G \to v$, $\quad \widehat{D} = \begin{cases} \emptyset, & \text{if } Models(D) = \mathcal{P}(\Sigma) \\ \bigcup_{I \in Min(D)} \{I \to v\} & \text{in other case} \end{cases}$

In the following theorem we prove that D and $\widehat{D}$ are semantically equivalent, that is, they have the same models.

Theorem 1. *For each Σ-interpretation I and each $Horn^{\supset}$ Σ-clause D it holds: $I \Vdash \widehat{D} \iff I \Vdash D$*

Proof. For $D = v$, the theorem is trivial. Let D be $G \to v$.

($\Longrightarrow$) Let us suppose that $I \not\Vdash D$. Then $v \notin I$. Since $I \not\Vdash D$, $Min(D)$ is not empty. Therefore there exists some $J \in Min(D)$ such that $J \subseteq I$ and $J \to v$ is a clause of the program $\widehat{D}$. Then $I \Vdash J$ and $v \notin I$ imply $I \not\Vdash \widehat{D}$.

($\Longleftarrow$) Let us suppose that $I \Vdash D$. If $v \in I$ then trivially $I \Vdash \widehat{D}$. If $v \notin I$ then $I \not\Vdash G$. Let $J \to v$ be a clause in the program $\widehat{D}$ for some $J \in Min(D)$ (if $\widehat{D}$ were empty then trivially $I \Vdash \widehat{D}$). If J were a (proper) subset of I, by persistence of goals we obtain $J \not\Vdash G$. But this implies $J \Vdash D$ which contradicts $J \in Min(D)$. That is, each $J \in Min(D)$ is not a subset of I and then trivially $I \Vdash J \to v$. Therefore $I \Vdash \widehat{D}$. ∎

Corollary 1. *Each $Horn^{\supset}$ Σ-program P is equivalent to a Horn Σ-program $\widehat{P}$.*

Now we are going to consider a concrete $Horn^{\supset}$ clause D whose $\widehat{D}$ needs to have an exponential number of clauses with respect to the symbols in D.

Lemma 4. *Let D be the following Σ-clause*

$$([(a_{11} \to b) \wedge (a_{12} \to b) \wedge \ldots \wedge (a_{1n} \to b)] \supset b \wedge$$

$$\ldots$$

$$[(a_{n1} \to b) \wedge (a_{n2} \to b) \wedge \ldots \wedge (a_{nn} \to b)] \supset b\ ) \to a$$

where $\Sigma = \{a_{ij} \mid i, j \in \{1, \ldots, n\}\} \cup \{b, a\}$. Each Σ-interpretation of the form $\{a_{1k_1}, a_{2k_2}, \ldots, a_{nk_n}\}$, with $k_j \in \{1, \ldots, n\}$ belongs to $Min(D)$.

Proof. Let I be one of such Σ-interpretations. Without loss of generality, let us suppose I to be $\{a_{11}, \ldots, a_{n1}\}$. The given clause D is $(G_1 \wedge \ldots \wedge G_n) \to a$ where each G_i is the goal $((a_{i1} \to b) \wedge \ldots \wedge (a_{in} \to b)) \supset b$. First let us prove that for each proper subset $J \subset I$, it holds that $J \Vdash D$. Since there exists some $a_{i1} \notin J$, then $J \Vdash (a_{i1} \to b) \wedge \ldots \wedge (a_{in} \to b)$ and $J \not\Vdash b$. Then $J \not\Vdash G_i$ and therefore $J \Vdash (G_1 \wedge \ldots \wedge G_n) \to a$. Now let us see that $I \not\Vdash D$. Since $a_{11} \in I$, for every Σ-interpretation K such that $I \subseteq K$ and $K \Vdash (a_{11} \to b) \wedge \ldots \wedge (a_{1n} \to b)$ it holds that $K \Vdash b$ and therefore $I \Vdash G_1$. Similarly, we can obtain $I \Vdash G_i$ for each $i \in \{1, \ldots, n\}$ and then, since $a \notin I$, $I \not\Vdash D$. ∎

The set of $Horn$ clauses $\widehat{D}$ obtained by Definition 7 from the $Horn^{\supset}$ program D in Lemma 4 contains at least these n^n clauses:

$$\{I \to a \mid I \text{ is } \{a_{1k_1}, a_{2k_2}, \ldots, a_{nk_n}\}, \text{ with } k_j \in \{1, \ldots, n\}\} \tag{1}$$

The next result shows that this set of $Horn$ clauses is non-redundant.

Lemma 5. *Any set of Horn clauses equivalent to (1) has at least n^n clauses.*

Proof. Denote by $I_r \to a$ the r-th clause in (1), for $1 \leq r \leq n^n$. I_r is not a model of the r-th clause in (1), but satisfies any other clause in (1). In addition, for each pair I_i, I_j with $1 \leq i \neq j \leq n^n$, the intersection $I_i \cap I_j$ is a model of (1).

Suppose that there exists a set H of $Horn$ clauses equivalent to (1) whose number of clauses is smaller than n^n. There must be at least two different Σ-interpretations I_i and I_j that falsify the same clause c in H. Since we are dealing with $Horn$ clauses, the interpretation $I_i \cap I_j$ falsifies c and therefore $I_i \cap I_j$ is not a model of H which is a contradiction. ∎

5 Boolean Formulas are pwm-reducible to $Horn^{\supset}$ programs

In this section we present a relative solution about the non-learnability of the class $Horn^{\supset}$. We show that if $Horn^{\supset}$ programs were learnable then boolean formulas would be also learnable. Namely, we show that a set of especial $Horn^{\supset}$ programs can simulate boolean formulas. The simulation is in the model of prediction with membership queries, where the notion of simulation is characterized by the concept of pwm-reduction.

The pwm-reduction from boolean formulas $\mathcal{C}_{BF}$ to $Horn^{\supset}$ is divided in two stages. Let Σ be a fixed signature. Let $GHorn^{\supset}$ (respectively $DHorn^{\supset}$) be the representation class whose elements are $\langle u, x\rangle$, where x is a Σ-interpretation and u is a Σ-goal G (respectively a Σ-clause D). First we prove that $GHorn^{\supset}$ is pwm-reducible to $DHorn^{\supset}$ and then we present a pwm-reduction from $\mathcal{C}_{BF}$ to $GHorn^{\supset}$.

Lemma 6. $GHorn^{\supset} \leq_{pwm} DHorn^{\supset}$

Proof. Let Σ be a fixed signature and b be a fixed variable that is not in Σ. Consider the representation class $GHorn^{\supset}$ over signature Σ and $DHorn^{\supset}$ over signature $\Sigma \cup \{b\}$. For each Σ-goal G, $G \rightarrow b$ is a $\Sigma \cup \{b\}$-clause. For any Σ-interpretation I it holds that $I \Vdash_{\Sigma} G \Longleftrightarrow I \not\Vdash_{\Sigma\cup\{b\}} (G \rightarrow b)$.

Formally we define functions g, h and f as follows. For all natural number s and for every concept representation G such that $|G| \leq s$, define $g(s, |\Sigma|, G) = G \rightarrow b$. For every Σ-interpretation I, define $f(s, |\Sigma|, I) = I$ and for every $\Sigma\cup\{b\}$-interpretation J define $h(s, |\Sigma|, J) = \top$ if $b \in J$ or $h(s, |\Sigma|, J) = J$ if $b \notin J$. These functions preserve conditions (1), (2)' and (3)' in Definition 1. ∎

The second step is based completely on the fact that monotone boolean circuits are as hard to predict as general boolean circuits. This result was proved in [6]. An exhaustive analysis of its proof allows us to ensure that monotone boolean formulas (denoted by $\mathcal{C}_{MBF}$) are as hard to predict as general ones.

Theorem 2. *[6]* $\mathcal{C}_{BF} \leq_{pwm} \mathcal{C}_{MBF}$

Next, we reduce the class of monotone boolean formulas to $GHorn^{\supset}$.

Theorem 3. $\mathcal{C}_{MBF} \leq_{pwm} GHorn^{\supset}$

Proof. Let C be a monotone boolean formula over signature Σ. From C we can construct, by induction, a goal C' over signature $\Sigma \cup \{b\}$, where $b \notin \Sigma$.

1. If C is a variable v, then C' is v
2. if C is of the form $C_1 \wedge \ldots \wedge C_k$, then C' is $C'_1 \wedge \ldots \wedge C'_k$
3. if C is of the form $C_1 \vee \ldots \vee C_k$, then C' is $((C'_1 \rightarrow b) \wedge \ldots \wedge (C'_k \rightarrow b)) \supset b$

The following proposition allows us to prove the theorem.

Proposition 2. *Given a signature Σ, a variable $b \notin \Sigma$, and a monotone boolean formula C over Σ: for all Σ-interpretation I, $I \Vdash_\Sigma C \Longleftrightarrow I \Vdash_{\Sigma\cup\{b\}} C'$*

Proof. The proof is again by induction over the structure of C. The nontrivial case is when $C = C_1 \vee \ldots \vee C_k$ and we suppose the proposition holds for $C_1, \ldots, C_k$. Let us see that $I \Vdash_\Sigma C_1 \vee \ldots \vee C_k \Longleftrightarrow I \Vdash_{\Sigma\cup\{b\}} ((C'_1 \rightarrow b) \wedge \ldots \wedge (C'_k \rightarrow b)) \supset b$

($\Longrightarrow$) Suppose $I \nVdash_{\Sigma\cup\{b\}} ((C'_1 \rightarrow b) \wedge \ldots \wedge (C'_k \rightarrow b)) \supset b$. There must exist J such that: $I \subseteq J$; for all $i \in \{1, \ldots, k\}$, $J \Vdash_{\Sigma\cup\{b\}} C'_i \rightarrow b$; and $J \nVdash_{\Sigma\cup\{b\}} b$. Therefore, for all $i \in \{1, \ldots, k\}$, $J \nVdash_{\Sigma\cup\{b\}} C'_i$. By persistence of goals, for all $i \in \{1, \ldots, k\}$, $I \nVdash_{\Sigma\cup\{b\}} C'_i$. Hence, by hypothesis of induction, for all $i \in \{1, \ldots, k\}$, $I \nVdash_\Sigma C_i$. Thus, $I \nVdash_\Sigma C_1 \vee \ldots \vee C_k$.

($\Longleftarrow$) Suppose $I \nVdash_\Sigma C_1 \vee \ldots \vee C_k$. That is, for all $i \in \{1, \ldots, k\}$, $I \nVdash_\Sigma C_i$. By hypothesis of induction, for all $i \in \{1, \ldots, k\}$, $I \nVdash_{\Sigma\cup\{b\}} C'_i$. Hence, for all $i \in \{1, \ldots, k\}$, $I \Vdash_{\Sigma\cup\{b\}} C'_i \rightarrow b$. Since $b \notin I$, $I \nVdash_{\Sigma\cup\{b\}} ((C'_1 \rightarrow b) \wedge \ldots \wedge (C'_k \rightarrow b)) \supset b$. ∎

Now we define functions g, h, and f. For all natural number s and for every monotone boolean formula C over Σ, such that $|C| \leq s$, let C' be the $\Sigma\cup\{b\}$-goal obtained from C as described above, where $b \notin \Sigma$. We define $g(s, |\Sigma|, C) = C'$. For every Σ-interpretation I, $f(s, |\Sigma|, I) = I$ and for every $\Sigma\cup\{b\}$-interpretation J, $h(s, |\Sigma|, J) = \top$ if $b \in J$ or $h(s, |\Sigma|, J) = J$ if $b \notin J$. These functions preserve conditions (1), (2) and (3) in Definition 1. ∎

Therefore, by Lemmas 2 and 3, if $DHorn^\supset$ is polynomially predictable with membership queries, then $\mathcal{C}_{BF}$ is polynomially predictable with membership queries. As a consequence, by Lemma 1, if $DHorn^\supset$ is polynomially learnable, then $\mathcal{C}_{BF}$ is polynomially predictable with membership queries. However, the well-known Angluin and Kharitonov's result below (see [3]) ensures us that predicting $\mathcal{C}_{BF}$ in polynomial time and using membership queries is a hard problem.

Theorem 4. *[3] If we assume the intractability of any of the following three problems: testing quadratic residues modulo a composite, inverting RSA encryption, or factoring Blum integers, then $\mathcal{C}_{BF}$ is not polynomially predictable with membership queries.*

6 Conclusions

In this paper, we have presented a pwm-reduction from boolean formulas to conjunctions of $Horn^\supset$ clauses. Consequently, this class is not predictable even

with membership queries under cryptographic assumptions. This result gives rise to other questions: Could we give a negative result about the learnability of the class? Could we improve the relative result presented in this paper by showing that conjunctions of $Horn^{\supset}$ clauses are pwm-equivalent to boolean formulas?

References

1. Angluin, D. *Queries and Concept Learning*. Machine Learning, volume 2(4): 319–342, (1988).
2. Angluin, D., Frazier, M. and Pitt, L. *Learning Conjunctions of Horn Clauses*. Machine Learning, volume 9: 147–164, (1992).
3. Angluin, D. and Kharitonov, M. *When won't Membership Queries Help?*. Journal of Computer and System Sciences, 50(2): 336–355, April 1995.
4. Arruabarrena, R., Lucio P. and Navarro, M. *A Strong Logic Programming View for Static Embedded Implications*. In: Proc. of FOSSACS'99, Springer-Verlag Lect. Notes in Comput. Sciences 1578: 56–72 (1999).
5. Bugliesi, M., Lamma, E. and Mello, P. *Modularity in Logic Programming*. Journal of Logic Programming, (19-20): 443–502, (1994).
6. Dalmau, V. *A Dichotomy Theorem for Learning Quantified Boolean Formulas*. Machine Learning, volume 35(3): 207–224 (1999).
7. Gabbay, D. M. *N-Prolog: An Extension of Prolog with Hypothetical Implications. II. Logical Foundations and Negation as Failure*. Journal of Logic Programming 2(4): 251–283 (1985).
8. Giordano, L., and Martelli, A. *Structuring Logic Programs: A Modal Approach*. Journal of Logic Programming 21: 59–94 (1994).
9. Giordano, L., Martelli, A., and Rossi, G. *Extending Horn Clause Logic with Implication Goals*. Theoretical Computer Science 95: 43–74, (1992).
10. Kearns, M. and Valiant, L. *Criptographic Limitations on Learning Boolean Formulae and Finite Automata*. Journal of the ACM 41(1): 67–95, (1994).
11. Lucio, P. *Structured Sequent Calculi for Combining Intuitionistic and Classical First-Order Logic*. In: Proc. of FroCoSS'2000,Springer-Verlag Lect. Notes in Artificial Intelligence 1794: 88–104 (2000).
12. Meseguer, J. *Multiparadigm Logic Programming*. In: Proc. of ALP'92, Springer-Verlag Lect. Notes in Comput. Sciences 632: 158–200, (1992).
13. Miller, D. *A Logical Analysis of Modules in Logic Programming*. In: Journal of Logic Programming 6: 79–108, (1989).
14. Miller, D., Nadathur, G., Pfenning, F. and Scedrov, A. *Uniform Proofs as a Foundation for Logic Programming*. Annals of Pure and App. Logic 51: 125–157, (1991).
15. Monteiro, L., Porto, A. *Contextual Logic Programming*. In: Proc. 6th International Conf. on Logic Programming 284–299, (1989).
16. Moscowitz, Y., and Shapiro, E. *Lexical Logic Programs*. In: Proc. 8th International Conf. on Logic Programming 349–363, (1991).
17. Navarro, M. *From Modular Horn Programs to Flat Ones: a Formal Proof for the Propositional Case*. In: Proc. of ISIICT 2004 (Int. Symp. on Innovation in Information and Communication Technology), Amman, Jordan. April 2004.
18. Pitt, L. and Warmuth, M.K. *Prediction-Preserving Reducibility*. Journal of Computer and System Sciences, 41(3): 430–467 (1990)
19. Valiant L.G. *A Theory of the Learnable*. Communications of the ACM, 27: 1134–1142 (1984)