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 $\text{Horn} \supset$ with respect to $\mathcal{FO}$. 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 $\mathcal{FO}$ (see also
[12,11]).

Our aim is to study the learnability of propositional $\text{Horn} \supset$ clauses since
this class of formulas, extending propositional $\text{Horn}$ 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 $\text{Horn} \supset$ (restricted to the propositional
case) is presented. In Section 4 we show that any model-based translation from
$\text{Horn} \supset$ clauses to equivalent $\text{Horn}$ clauses obtains, in general, an exponential
number of clauses. This result suggests us that finding a polynomial-time algo-

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

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 $C$ 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 $K_C(v)$ 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 $K_C(h) = c$ and a counterexample, in the symmetric difference of $c$ and $K_C(h)$, 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 $C$, 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 $c = K_C(u)$, $A$ outputs a concept name $v$ such that $c = K_C(v)$.

A representation of concepts $C$ is polynomially learnable if there is a learning algorithm $A$ that runs in polynomial time and exactly learns $C$. 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 $\epsilon$. 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 $D$ 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 $D$ 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 $D$.

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 $C$ if and only if for all positive integers $s$ and $n$, for all positive rational $\epsilon$, for all concept name $u \in S^*$ with $|u| \leq s$, when $A$ runs with inputs $s$, $n$, and $\epsilon$, and a target concept $c = K_C(u)$ and $D$, $A$ 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 $\epsilon$.

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

**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 \textit{pwm-reducibility} as defined in [3]. It is denoted by \( \leq_{pwm} \).

**Definition 1.** Let \( C \) and \( C' \) be representations of concepts. Let \( \top \) and \( \bot \) be two different symbols not occurring in \( S \). Then \( C \) is \( \text{pwm-reducible to } C' \), 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 \( x \in K_C(u) \) if and only if \( x' \in K_C(g(s, n, u)) \).

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 \( x' \in K_C(g(s, n, u)) \); if \( h(s, n, x') = \bot \) then \( x' \notin K_C(g(s, n, u)) \); and otherwise \( x' \in K_C(g(s, n, u)) \) if and only if \( x \in K_C(u) \). 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 “\( x \in K_C(u) \)” can be replaced with “\( x \notin K_C(u) \)”, 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 \( \text{pwm-reduction is transitive, i.e., let } C, C', \text{ and } C'' \text{ be representations of concepts, if } C \leq_{pwm} C' \leq_{pwm} C'' \text{ then } C \leq_{pwm} C'' \).

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

3 The programming language \textit{Horn}⊃

In this section we introduce the programming language \textit{Horn}⊃ 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) \textit{Horn} clause language by adding the intuitionistic implication \( \supset \) in goals and clause bodies. Let \( \Sigma \) be a set of propositional variables (or signature). The \( \Sigma \)-clauses, named \( D \), and the \( \Sigma \)-goals, named \( G \), are recursively defined as follows (where \( v \) stands for any propositional variable in \( \Sigma \)):
A Horn\(\supset\) \(\Sigma\)-program is a finite set (or conjunction) of \(\Sigma\)-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 \(\Sigma = \{a, b, c, d\}\)

\[
\{(b \rightarrow c) \supset a, b, ((a \land (b \rightarrow c)) \supset ((b \rightarrow c) \land (a \rightarrow d)) \supset a) \rightarrow d\}
\]

The second program clause is simply \(b\). The first and the third program clauses are of the form \(G \rightarrow v\). In the first clause, the goal \(G\) is \((b \rightarrow c) \supset c\). 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 \(D_1 = a \land (b \rightarrow c)\), \(D_2 = (b \rightarrow c) \land (a \rightarrow d)\), and \(G_3 = a\). \(D_1\) and \(D_2\) are local sets with two clauses and they can also be written \(D_1 = \{a, (b \rightarrow c)\}\) and \(D_2 = \{(b \rightarrow c), (a \rightarrow d)\}\) 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 \(\Sigma\), the model semantics for the propositional Horn\(\supset\) is given by the set of Kripke \(\Sigma\)-structures \(\text{Mod}(\Sigma) = \{K_I \mid I \subseteq \Sigma\}\) where \(K_I\) denotes the partially ordered set of worlds \(\{J \subseteq \Sigma \mid I \subseteq J\}\).

Well-formed \(\Sigma\)-formulas are built, from propositional variables in \(\Sigma\), using classical connectives (\(\neg, \land, \lor\) and \(\rightarrow\)) and the intuitionistic implication (\(\supset\)). The satisfaction relation \(\models_{\Sigma}\) between a \(\Sigma\)-structure \(K_I\) and a \(\Sigma\)-formula \(\varphi\) requires the formula \(\varphi\) 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 \(K_I \in \text{Mod}(\Sigma)\) and \(\varphi\) a \(\Sigma\)-formula. We say that

(a) \(K_I, I \models_{\Sigma} \varphi\) (\(K_I\) satisfies \(\varphi\)) iff \(I \models_{\Sigma} \varphi\) (\(\varphi\) is forced in \(I\))

(b) The binary forcing relation \(\models_{\Sigma}\) (or simply \(\models\) if there is no confusion about the signature) is inductively defined as follows:

\[
\begin{align*}
I \models \text{False} \\
I \models v \text{ if } v \in I \text{ for } v \in \Sigma \\
I \models \neg \varphi \text{ if } I \not\models \varphi \\
I \models \varphi \land \psi \text{ if } I \models \varphi \text{ and } I \models \psi \\
I \models \varphi \lor \psi \text{ if } I \models \varphi \text{ or } I \models \psi \\
I \models \varphi \rightarrow \psi \text{ if } I \models \varphi \text{ then } I \models \psi \\
I \models \varphi \supset \psi \text{ iff for all } J \subseteq \Sigma \text{ such that } I \subseteq J \text{ if } J \models \varphi \text{ then } J \models \psi
\end{align*}
\]

**Example 2.** Let \(\varphi\) be the formula (in this case a goal) \(((a \land c) \rightarrow b) \supset (c \land b)\).

\(I \models \varphi\) for \(I = \{a, b, c\}\), \(I = \{a, c\}\) and \(I = \{b, c\}\). \(I \not\models \varphi\) for \(I = \{a, b\}\) and \(I = \emptyset\). Note, for instance, that \(\{a, b\} \models (a \land c) \rightarrow b\) and \(\{a, b\} \not\models (c \land b)\).
This forcing relation does not behave monotonically with respect to the world ordering for general formulas. For instance, \( a \rightarrow b \) 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 \( \Sigma \)-formula \( \varphi \) is persistent when for any \( \Sigma \)-interpretation \( I \), if \( I \models \varphi \) then \( J \models \varphi \) for any \( \Sigma \)-interpretation \( J \) such that \( I \subseteq J \).

From the following proposition we obtain, in particular, that any \( \Sigma \)-goal \( G \) is a persistent formula.

**Proposition 1.** [4] Any \( v \in \Sigma \) is persistent. Any formula \( \varphi \supset \psi \) is persistent. If \( \varphi \) and \( \psi \) are persistent then \( \varphi \lor \psi \) and \( \varphi \land \psi \) are persistent.

**Definition 5.** Two formulas \( \varphi \) and \( \psi \) are semantically equivalent if both have the same meaning in each structure in \( \text{Mod}(\Sigma) \). In other words, if both are forced in the same \( \Sigma \)-interpretations.

### 4 Trying to learn \( \text{Horn}^\supset \) programs with \( \text{Horn} \) programs

A way to learn a \( \text{Horn}^\supset \) program \( P \) might consist on using the well-known algorithm of Angluin, Frazier and Pitt [2] for learning a conjunction of \( \text{Horn} \) 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 \( \text{Horn}^\supset \Sigma \)-program \( P \) to an equivalent \( \text{Horn} \Sigma \)-program \( \hat{P} \) yields a number of clauses that is exponential in the size of \( P \).

**Definition 6.** For each \( \text{Horn}^\supset \Sigma \)-clause \( D \), let \( \text{Models}(D) \) be the set \( \{ I \subseteq \Sigma \mid I \models D \} \). Let \( \text{Min}(D) \) be the set \( \{ I \subseteq \Sigma \mid I \not\models D \text{ but } J \models D, \text{ for all } J \subset I \} \). That is, \( \text{Min}(D) \) 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 \( \text{Horn}^\supset \Sigma \)-clause \( D \), the \( \text{Horn} \Sigma \)-program \( \hat{D} \) is defined as follows:

\[
\begin{align*}
\text{For } D &= v, \quad \hat{D} = \{ v \} \\
\text{For } D &= G \rightarrow v, \quad \hat{D} = \begin{cases} \\
\emptyset, \quad &\text{if } \text{Models}(D) = \mathcal{P}(\Sigma) \\
\bigcup_{I \in \text{Min}(D)} \{ I \rightarrow v \}, \quad &\text{in other case}
\end{cases}
\end{align*}
\]

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

**Theorem 1.** For each \( \Sigma \)-interpretation \( I \) and each \( \text{Horn}^\supset \Sigma \)-clause \( D \) it holds:

\( I \models \hat{D} \iff I \models D \)

**Proof.** For \( D = v \), the theorem is trivial. Let \( D = G \rightarrow v \).
(\(\implies\)) Let us suppose that \(I \not\models D\). Then \(v \notin I\). Since \(I \not\models D\), \(\text{Min}(D)\) is not empty. Therefore there exists some \(J \in \text{Min}(D)\) such that \(J \subseteq I\) and \(J \models v\) is a clause of the program \(\hat{D}\). Then \(I \models J\) and \(v \notin I\) imply \(I \not\models \hat{D}\).

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

**Corollary 1.** Each \(\text{Horn}^\to\Sigma\)-program \(P\) is equivalent to a \(\text{Horn}\Sigma\)-program \(\hat{P}\).

Now we are going to consider a concrete \(\text{Horn}^\to\Sigma\)-clause \(D\) whose \(\hat{D}\) needs to have an exponential number of clauses with respect to the symbols in \(D\).

**Lemma 4.** Let \(D\) be the following \(\Sigma\)-clause

\[
\left(\left(\left(\left(a_{11} \to b\right) \land \left(a_{12} \to b\right) \land \ldots \land \left(a_{1n} \to b\right)\right) \supset b \land \ldots \left(\left(a_{n1} \to b\right) \land \left(a_{n2} \to b\right) \land \ldots \land \left(a_{nn} \to b\right)\right) \supset b \right) \to a
\]

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

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

The set of \(\text{Horn}\) clauses \(\hat{D}\) obtained by Definition 7 from the \(\text{Horn}^\to\Sigma\) program \(D\) in Lemma 4 contains at least these \(n^n\) clauses:

\[
\{I \to a \mid I \models \{a_{1k_1}, a_{2k_2}, \ldots, a_{nk_n}\}, \text{ with } k_j \in \{1, \ldots, n\}\} \quad (1)
\]

The next result shows that this set of \(\text{Horn}\) clauses is non-redundant.

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

**Proof.** Denote by \(I_v \to a\) the \(r\)-th clause in \((1)\), for \(1 \leq r \leq n^n\). \(I_v\) 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 $\Sigma$-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.

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

In this section we present a relative solution about the non-learnability of the class $\text{Horn}^\supset$. We show that if $\text{Horn}^\supset$ programs were learnable then boolean formulas would be also learnable. Namely, we show that a set of especial $\text{Horn}^\supset$ 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 $C_{BF}$ to $\text{Horn}^\supset$ is divided in two stages. Let $\Sigma$ be a fixed signature. Let $\text{GHorn}^\supset$ (respectively $\text{DHorn}^\supset$) be the representation class whose elements are $(u,x)$, where $x$ is a $\Sigma$-interpretation and $u$ is a $\Sigma$-goal $G$ (respectively a $\Sigma$-clause $D$). First we prove that $\text{GHorn}^\supset$ is pwm-reducible to $\text{DHorn}^\supset$ and then we present a pwm-reduction from $C_{BF}$ to $\text{GHorn}^\supset$.

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

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

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 $\Sigma$-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 $C_{MBF}$) are as hard to predict as general ones.

\textbf{Theorem 2.} [6] $C_{BF} \leq_{\text{pwm}} C_{MBF}$

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

\textbf{Theorem 3.} $C_{MBF} \leq_{\text{pwm}} \text{GHorn}^\supset$

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

In this paper, we have presented a pwm-reduction from boolean formulas to conjunctions of Horn 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 $\text{Horn} \supset$ clauses are pwm-equivalent to boolean formulas?

References