Boolean Formulas are Hard to Learn for Most Gate Bases

Víctor Dalmau *

Departament LSI, Universitat Politècnica de Catalunya,
Mòdul C5. Jordi Girona Salgado 1-3. Barcelona 08034, Spain,
`dalmau@lsi.upc.es`

Abstract

Boolean formulas are known not to be PAC-predictable even with membership queries under some cryptographic assumptions. In this paper, we study the learning complexity of some subclasses of boolean formulas obtained by varying the basis of elementary operations allowed as connectives. This broad family of classes includes, as a particular case, general boolean formulas, by considering the basis given by $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$. We completely solve the problem. We prove the following *dichotomy theorem*: For any set of basic boolean functions, the resulting set of formulas is either polynomially learnable from equivalence queries or membership queries alone or else it is not PAC-predictable even with membership queries under cryptographic assumptions. We identify precisely which sets of basic functions are in which of the two cases. Furthermore, we prove than the learning complexity of formulas over a basis depends only on the absolute expressivity power of the class, ie., the set of functions that can be represented regardless of the size of the representation. In consequence, the same classification holds for the learnability of boolean circuits.

*This work has been partly supported by: the Spanish Government through the DGES under project PB95-0787 (KOALA), the Catalan Government through CIRIT under the project 1997SGR-00366 (SGR), and the European Commission through the ESPRIT BRA Program under the following projects: Working Group 27150 (NeuroCOLT2), project 20244 (ALCOM IT) and project TIC97-1475-CE (CICYT).

1 Introduction

The problem of learning an unknown boolean formula under some determined protocol has been widely studied. It is well known that, even restricted to propositional formulas, the problem is hard [4, 18] in the usual learning models. Therefore researchers have attempted to learn subclasses of propositional boolean formulas obtained by enforcing some restrictions on the structure of the formula, specially subclasses of boolean formulas in disjunctive normal form (DNF).

In this paper we take a different approach. We study the complexity of learning subclasses of boolean formulas obtained placing some restrictions in the elementary boolean functions that can be used to build the formulas. In general, boolean formulas are constructed by using elementary functions from a complete basis, generally $\{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$. In this paper we will allow formulas to use as a basis any arbitrary set of boolean functions.

More precisely, let $F = \{f_1, \ldots, f_m\}$ be a finite set of boolean functions. A formula in $\mathrm{FOR}(F)$ can be any of (a) a boolean variable, or (b) an expression of the form $f(g_1, \ldots, g_k)$ where f is a k-ary function in F and $g_1, \ldots, g_k$ are formulas in $\mathrm{FOR}(F)$.

For example, consider the problem of learning a monotone boolean formula. Every such formula can be expressed as a formula in the class $\mathrm{FOR}(\{\mathrm{AND}, \mathrm{OR}\})$. The main result of this paper characterizes the complexity of learning $\mathrm{FOR}(F)$ for every finite set F of boolean functions. The most striking feature of this characterization is that for any F, $\mathrm{FOR}(F)$ is either polynomially learnable with equivalence or membership queries alone or, under some cryptographic assumptions, not polynomially predictable even with membership queries.

This dichotomy is somewhat surprising since one might expect that any such large and diverse family of concept classes would include some representatives of the many intermediate learning models such as exact learning with equivalence and membership queries, PAC learning with and without membership queries and PAC-prediction without membership queries.

Furthermore, we give an interesting classification of the polynomially learnable classes. We show that, in a sense that will be made precise later, $\mathrm{FOR}(F)$ is polynomially learnable if and only if at least one of the following conditions holds:

(a) Every function $f(x_1, x_2, \ldots, x_n)$ in F is definable by an expression of the form $c_0 \vee (c_1 \wedge x_1) \vee (c_2 \wedge x_2) \vee \cdots \vee (c_n \wedge x_n)$ for some boolean coefficients c_i $(1 \leq i \leq n)$.

(b) Every function $f(x_1, x_2, \ldots, x_n)$ in F is definable by an expression of the form $c_0 \wedge (c_1 \vee x_1) \wedge (c_2 \vee x_2) \wedge \cdots \wedge (c_n \vee x_n)$ for some boolean coefficients c_i $(1 \leq i \leq n)$.

(a) Every function $f(x_1, x_2, \ldots, x_n)$ in F is definable by an expression of the form $c_0 \oplus (c_1 \wedge x_1) \oplus (c_2 \wedge x_2) \oplus \cdots \oplus (c_n \wedge x_n)$ for some boolean coefficients c_i $(1 \leq i \leq n)$.

There is another rather special feature of this result. Learnability of boolean formulas over a basis F depends only on the set of functions that can be expressed as a formula in $\mathrm{FOR}(F)$ but it does not depend on the size of the representation. As a consequence of this fact, the same dichotomy holds for representation classes which, in terms of absolute expressivity power, are equivalent to formulas, such as boolean circuits.

As an intermediate tool for our study we introduce a link with some well known algebraic structure, called *clones* in Universal Algebra. In particular, we make use of very remarkable result on the structure of boolean functions proved by Post [21]. This approach has been successful in the study of other computational problems such as satisfiability, tautology and counting problems of boolean formulas [23], learnability of quantified formulas [9] and Constraint Satisfaction Problems [13, 14, 15, 16, 12].

Finally we mention some similar results: a dichotomy result for satisfiability, tautology and some counting problems over closed sets of boolean functions [23], the circuit value problem [11, 10], the satisfiability of generalized formulas [24], the inverse generalized satisfiability problem [17], the generalized satisfiability counting problem [7], the approximability of minimization and maximization problems [6, 19, 20], the optimal assignments of Generalized Propositional Formulas [22] and the learnability of quantified boolean formulas [8].

2 Learning Preliminaries

Most of the terminology about learning comes from [4]. Strings over $X = D^*$ will represent both examples and concept names. A *representation of concepts* $\mathcal{C}$ is any subset of $X \times X$. We interpret an element $\langle u, x\rangle$ of $X \times X$ as consisting of a *concept name* u and an *example* x. The example x is a member of the concept u if and only if $\langle u, x\rangle \in \mathcal{C}$.

Define the *concept represented by* u as $K_{\mathcal{C}}(u) = \{x : \langle u, x\rangle \in \mathcal{C}\}$. The *set of concepts represented by* $\mathcal{C}$ is $K_{\mathcal{C}} = \{K_{\mathcal{C}}(u) : u \in X\}$.

Along these pages we use two models of learning, all of them fairly standard: Angluin's model of exact learning with queries defined by Angluin [1], and the model of PAC-prediction with membership queries as defined by Angluin and Kharitonov [4].

To compare the difficulty of learning problems in the prediction model we use a slight generalization of the *prediction-preserving reducibility with membership queries* [4].

Definition 1 *Let $\mathcal{C}$ and $\mathcal{C}'$ be representations of concepts. Let $\perp$ and $\top$ be elements not in X. Then $\mathcal{C}$ is* pwm-reducible *to $\mathcal{C}'$, denoted $\mathcal{C} \leq_{pwm} \mathcal{C}'$, if and only if there exist four mappings g, f, h, and j with the following properties:*

1. *There is a nondecreasing polynomial q such that for all natural numbers s and n and for $u \in X$ 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 string $u \in X$ with $|u| \leq s$, and for every $x \in X$ with $|x| \leq n$, $f(s,n,x)$ is a string x' and $x \in K_{\mathcal{C}}(u)$ if and only if $x' \in K_{\mathcal{C}'}(g(s,n,u))$. Moreover, f is computable in time bounded by a polynomial in s, n, and $|x|$, hence there exists a nondecreasing polynomial t such that $|x'| \leq t(s,n,|x|)$.*
3. *For all natural numbers s and n, for every string $u \in X$ with $|u| \leq s$, for every $x' \in X$, and for every $b \in \{\top, \perp\}$, $h(s,n,x')$ is a string $x \in X$, and $j(s,n,x',b)$ is either $\perp$ or $\top$. Furthermore $x' \in K_{\mathcal{C}'}(g(s,n,u))$ if and only if $j(s,n,x',b) = \top$, where $b = \top$ if $x \in K_{\mathcal{C}}(u)$ and $b = \perp$ otherwise. Moreover, h and j are computable in time bounded by a polynomial in s, n, and $|x'|$.*

In (2), and independently in (3), the expression "$x \in K_{\mathcal{C}}(u)$" can be replaced with "$x \notin K_{\mathcal{C}}(u)$", as discussed in [4].

The following results are obtained adapting slightly some proofs in [4].

Lemma 1 *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 2 *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 Clones

Let D be finite set called *domain*. An n-adic *function* over D is a map $f : D^n \longrightarrow D$. Let $\mathcal{F}^D$ be the set of all the functions over the domain D. Let $\mathcal{P}$ be a class of functions over D. Let $\mathcal{P}_n$ be the n-adic functions in $\mathcal{P}$. We shall say that $\mathcal{P}$ is a clone if it satisfies the following conditions:

C_1 For each $n \geq m \geq 1$, $\mathcal{P}$ contains the *projection function* $\mathrm{proj}_{n,m}$, defined by

$$\mathrm{proj}_{n,m}(x_1, \ldots, x_n) = x_m$$

C_2 For each $n, m \geq 1$, each $f \in \mathcal{P}_n$ and each $g_1, \ldots, g_n \in \mathcal{P}_m$. $\mathcal{P}_m$ contains the composite function $h = f[g_1, \ldots, g_n]$ defined by

$$h(x_1, \ldots, x_m) = f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m))$$

If $F \subseteq \mathcal{F}^D$ is any set of functions over D, there is a smallest clone containing all of the functions in F; this is the clone generated by F, and we denote it $\langle F \rangle$ If $F = \{f_1, \ldots, f_k\}$ is a finite set, we may write $\langle f_1, \ldots, f_k \rangle$ for $\langle F \rangle$, and refer to $f_1, \ldots, f_k$ as generators of $\langle F \rangle$. The set of clones over a finite domain D is closed under intersection and therefore it constitutes a lattice, with meet ($\wedge$) and join ($\vee$) operations defined by:

$$\bigwedge_{i \in I} C_i = \bigcap_{i \in I} C_i \qquad \bigvee_{i \in I} C_i = \left\langle \bigcup_{i \in I} C_i \right\rangle$$

There is a smallest clone which is the intersection of all clones; we shall denote it $\mathcal{I}_D$. It is easy to see that $\mathcal{I}_D$ contains exactly the projections over D.

3.1 Boolean Case

Operations on a 2-element set, say $D = \{0, 1\}$ are boolean operations. The lattice of the clones over the boolean domain was studied by Post, leading to a full description [21]. The proof is too long to be included here. We will give only the description of the lattice.

A usual way to describe a poset $(P, \leq)$ (and a lattice in particular) is by depicting a *diagram* with the relation *coverage*, where for every $a, b \in P$ we say that a covers b if $b \leq a$ and if c is an element in P such that $b \leq c \leq a$, then either $c = a$ or $c = b$. The diagram of the

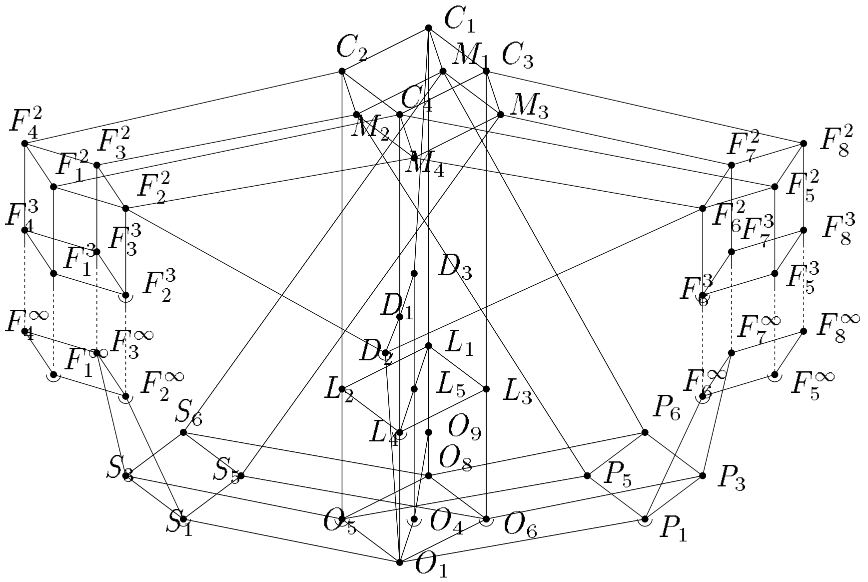

Figure 1: Post Lattice

lattice of clones on $D = \{0, 1\}$, often called *Post's lattice* is depicted in Figure 1. The clones are labeled according their standard names.

A clone C is *join irreducible* iff $C = C_1 \vee C_2$ always implies $C = C_1$ or $C = C_2$. In Figure 1, the join irreducible clones of the diagram are denoted by $\smile$. Since $\langle \mathcal{P} \rangle = \bigvee_{f \in \mathcal{P}} \langle f \rangle$, it follows that the join irreducible clones are generated by a single operation, furthermore, it suffices to present a generating operation for each join irreducible clone of Post's lattice. Table 2 associates to every join irreducible clone $\mathcal{C}$, its generating operation $\varphi_{\mathcal{C}}$.

For a n-ary boolean function f define its *dual* $\mathrm{dual}(f)$ by

$$\mathrm{dual}(f)(x_1, \ldots, x_n) = \neg f(\neg x_1, \ldots, \neg x_n).$$

Obviously, $\mathrm{dual}(\mathrm{dual}(f)) = f$. Furthermore, f is self-dual iff $\mathrm{dual}(f) = f$. For a class F of boolean functions define $\mathrm{dual}(F) = \{\mathrm{dual}(f) : f \in F\}$. The classes F and $\mathrm{dual}(F)$ are called dual. Notice that $\langle \mathrm{dual}(F) \rangle = \mathrm{dual}(\langle F \rangle)$.

D_2	$\varphi_{D_2}(x,y,z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$
F_1^∞	$\varphi_{F_1^\infty}(x,y,z) = x \vee (y \wedge \bar{z})$
F_2^∞	$\varphi_{F_2^\infty}(x,y,z) = x \vee (y \wedge z)$
F_2^i $(i \geq 2)$	$\varphi_{F_2^i}(x_1,\ldots,x_{i+1}) = \bigwedge_{j=1}^{i+1}(x_1 \vee \cdots \vee x_{j-1} \vee x_{j+1} \vee \cdots \vee x_{i+1})$
F_5^∞	$\varphi_{F_5^\infty}(x,y,z) = x \wedge (y \vee \bar{z})$
F_6^∞	$\varphi_{F_6^\infty}(x,y,z) = x \wedge (y \vee z)$
F_6^i $(i \geq 2)$	$\varphi_{F_6^i}(x_1,\ldots,x_{i+1}) = \bigvee_{j=1}^{i+1}(x_1 \wedge \cdots \wedge x_{j-1} \wedge x_{j+1} \wedge \cdots \wedge x_{i+1})$
L_4	$\varphi_{L_4}(x,y,z) = x \oplus y \oplus z$
O_1	$\emptyset$
O_4	$\varphi_{O_4}(x) = \bar{x}$
O_5	$\varphi_{O_5}(x) = 1$
O_6	$\varphi_{O_6}(x) = 0$
P_1	$\varphi_{P_1}(x,y) = x \wedge y$
S_1	$\varphi_{S_1}(x,y) = x \vee y$

Figure 2: Generating operations for meet irreducible clones

4 The Dichotomy Theorem

A *base* F is a finite set of boolean functions $\{f_1, f_2, \ldots, f_n\}$ (f_i $1 \leq i \leq n$, denotes both the function and its symbol). We follow the standard definitions of boolean circuits and boolean formulas. The class of boolean circuits over the basis F, denoted by $\mathrm{CIR}(F)$, is defined to be the set of all the boolean circuits where every gate is a function in F. The class of boolean formulas over the basis F, denoted by $\mathrm{FOR}(F)$ is the class of circuits in $\mathrm{CIR}(F)$ with fan-out ≤ 1.

Given a boolean circuit C over the input variables $x_1, x_2, \ldots, x_n$, we denote by $[C]$ the function computed by C when the variables are taken as arguments in lexicographical order. In the same way, given a finite set of boolean functions F we define $[\mathrm{CIR}(F)]$ as the class of functions computed by circuits in $\mathrm{CIR}(F)$. Similarly, given a boolean formula Φ over the variables $x_1, x_2, \ldots, x_n$, we denote by $[\Phi]$ the function computed by Φ when the variables are taken as arguments in lexicographical order. In the same way, given a finite set of boolean functions F we define $[\mathrm{FOR}(F)]$ as the class of functions computed by formulas in $\mathrm{FOR}(F)$. Given a boolean circuit $C \in \mathrm{CIR}(F)$, it is possible to construct a formula $\Phi \in \mathrm{FOR}(F)$ computing the same function than C (the size of Φ can be exponentially bigger than the size of C but we are not concerned with the size of the representation). Thus $[\mathrm{FOR}(F)] = [\mathrm{CIR}(F)]$.

In fact, it is direct to verify that the class of functions $[\mathrm{CIR}(F)]$ contains the projections and it is closed under composition. Therefore, it constitutes a clone. More precisely, $[\mathrm{CIR}(F)]$ is exactly the clone generated by F, in our terminology

$$[\mathrm{CIR}(F)] = [\mathrm{FOR}(F)] = \langle F \rangle, \text{ for all bases } F.$$

The use of clone theory to study computational problems over boolean formulas was introduced in [23] studying the complexity of some computational problems over boolean formulas, such as satisfiability, tautology and some counting problems.

We say that an n-ary function f is disjunctive if there exists some boolean coefficients c_i $(0 \leq i \leq n)$ such that

$$f(x_1, \ldots, x_n) = c_0 \vee (c_1 \wedge x_1) \vee (c_2 \wedge x_2) \vee \cdots \vee (c_n \wedge x_n).$$

Similarly, we say that an n-ary function f is conjunctive if there exists some boolean coefficients c_i $(0 \leq i \leq n)$ such that

$$f(x_1, \ldots, x_n) = c_0 \wedge (c_1 \vee x_1) \wedge (c_2 \vee x_2) \wedge \cdots \wedge (c_n \vee x_n).$$

Accordingly, we say that an n-ary function f is linear if there exists some boolean coefficients c_i $(0 \leq i \leq n)$ such that

$$f(x_1, \ldots, x_n) = c_0 \oplus (c_1 \wedge x_1) \oplus (c_2 \wedge x_2) \oplus \cdots \oplus (c_n \wedge x_n).$$

Let F be a set of boolean functions. We will say that F is disjunctive (resp. conjunctive, linear) iff every function in F is disjunctive (resp. conjunctive, linear). We will say that F is basic iff F is disjunctive, conjunctive or linear.

For any set of boolean formulas (circuits) $\mathcal{B}$, we define $\mathcal{C}_{\mathcal{B}}$ as the representation of concepts formed from formulas (circuits) in $\mathcal{B}$. More precisely, $\mathcal{C}_{\mathcal{B}}$ contains all the tuples of the form $\langle C, x \rangle$ where C represents a formula (circuit) in $\mathcal{B}$ and x is a model satisfying C.

In this section we state and prove the main result of the paper.

Theorem 1 *(Dichotomy Theorem for the Learnability of Boolean Circuits and Boolean Formulas) Let F be a finite set of boolean functions. If F is basic, then $\mathcal{C}_{\mathrm{CIR}(F)}$ is both polynomially exactly learnable with $n+1$ equivalence queries, and polynomially exactly learnable* *with $n+1$ membership queries. Otherwise, $\mathcal{C}_{\mathrm{FOR}(F)}$ is not polynomially predictable with membership queries under the assumption that any of the following three problems are intractable: testing quadratic residues modulo a composite, inverting RSA encryption, or factoring Blum integers.*

We refer the reader to Angluin and Kharitonov [4] for definitions of the cryptographic concepts. We mention that the non-learnability results for boolean circuits hold also with the weaker assumption of the existence of public-key encryption systems secure against CC-attack, as discussed in [4].

Proof of Theorem 1

Learnability of formulas over disjunctive, conjunctive and linear bases is rather straightforward. Notice that if a basis F is disjunctive (resp. conjunctive, linear) then every function computed by a circuit in $\mathrm{CIR}(F)$ is also disjunctive (resp. conjunctive, linear). Thus, learning a boolean circuit over a basic basis is reduced to finding the boolean coefficients of the canonical expression. It is easy to verify that this task can be done in polynomial time with $n+1$ equivalence queries or $n+1$ membership queries, as stated in the theorem.

Now, we study which portion of Post's lattice is covered by these cases. Consider clone L_1 with generating set $L_1 = \langle \varphi_{O_4}, \varphi_{O_5}, \varphi_{O_6}, \varphi_{L_4} \rangle$. Since all the operations in the generating set of L_1 are linear, then the clone L_1 is linear. Similarly, clone P_6, generated by operations φ_{O_5}, φ_{O_6} and φ_{P_1} is conjunctive, since so is every operation in the generating set. Finally clone S_6, generated by operations φ_{O_5}, φ_{O_6} and φ_{S_1} is disjunctive, since so is every operation in the generating set. Thus, every clone contained in L_1, P_6, or S_6 is linear, conjunctive or disjunctive respectively. Actually, clone L_1 (resp. P_6, S_6) has been chosen carefully among the linear (resp. conjunctive, disjunctive) clones. It corresponds to the maximal linear (resp. conjunctive, disjunctive) clone in Post's lattice. It has been obtained as the join of all the linear (resp. conjunctive, disjunctive) clones and, in consequence, has the property that every linear (resp. conjunctive, disjunctive) clone is contained in L_1 (resp. P_6, S_6).

Let us study non-basic clones. With a simple inspection of Post's lattice we can infer that any clone not contained in L_1, S_6 or P_6 contains any of the following operations: $\varphi_{F_2^\infty}$, $\varphi_{F_6^\infty}$, φ_{D_2}. In Section 4.1

it is proved that if $\langle F \rangle$ contains any of the previous functions, then the class FOR(F) is not PAC-predictable even with membership queries under the assumptions of the theorem. ∎

4.1 Three Fundamental Non-Learnable Functions

Let $\mathcal{B} = \{\text{AND}, \text{OR}, \text{NOT}\}$ be the usual complete basis for boolean formulas. In [4], it is proved that $\mathcal{C}_{\text{FOR}(\mathcal{B})}$ is not polynomially predictable under the assumptions of Theorem 1. In this section we generalize the previous result to all bases F able to "simulate" any of the following three basic non-learnable functions: $\varphi_{F_2^\infty}$, $\varphi_{F_6^\infty}$, and φ_{D_2}. This three functions can be regarded as the basic causes for non-learnability in formulas.

The technique used to prove non-learnability results is a two-stage pwm-reduction from FOR($\mathcal{B}$). First, we prove as an intermediate result that monotone boolean formulas are as hard to learn, as general boolean formulas.

Lemma 3 *The class $\mathcal{C}_{\text{FOR}(\mathcal{B})}$ is pwm-reducible to $\mathcal{C}_{\text{FOR}(\{\text{AND},\text{OR}\})}$.*

Proof. Let Φ be a boolean formula with $x_1, \ldots, x_n$ as input variables. We can assume that NOT operations are applied only to input variables. Otherwise by using repeatedly De Morgan's law we can move every NOT function towards the input variables.

Consider the formula Ψ with $x_1, \ldots, x_n, y_1, \ldots, y_n$ as input variables, obtained modifying slightly formula Φ as follows: replace every NOT function applied to variable x_i, by the new input variable y_i. Finally, we define the formula Υ with $x_1, \ldots, x_n, y_1, \ldots, y_n$ as input variables to be:

$$\Upsilon(x_1, \ldots, x_n, y_1, \ldots, y_n) = \left(\Psi(x_1, \ldots, x_n, y_1, \ldots, y_n) \vee \bigvee_{1 \le i \le n} (x_i \wedge y_i) \right) \wedge \wedge \bigwedge_{1 \le i \le n} (x_i \vee y_i)$$

Formula Υ evaluates the following function:

$$\Upsilon(x_1, \ldots, x_n, y_1, \ldots, y_n) = \begin{cases} \Phi(x_1, \ldots, x_n) & \text{if } \forall i : 1 \le i \le n,\ x_i \neq y_i \\ 0 & \text{if } \exists i : 1 \le i \le n,\ x_i = y_i = 0 \\ 1 & \text{otherwise} \end{cases}$$

For every natural number s and for every concept representation $u \in \mathcal{C}_{\mathrm{FOR}(\mathcal{B})}$ such that $|u| \leq s$, let Φ be the boolean formula with n input variables represented by u, let u' be the representation of the monotone boolean formula Υ obtained from Φ as described above. We define $g(s, n, u) = u'$. For every assignments x and y of length n we define $f(s, n, x) = x\overline{x}$, $h(s, n, xy) = x$ and

$$j(s, n, xy, b) = \begin{cases} b & \text{if } x = \overline{y} \\ \perp & \text{if } \exists i : 1 \leq i \leq n, x_i = y_i = 0 \\ \top & \text{otherwise} \end{cases}$$

Clearly, f, g, h and j satisfy the conditions (1), (2) and (3) in Definition 1 and therefore define a pwm-reduction.

Technical note: In the proof of this prediction with membership reduction and in the next ones, functions f, g, h, j have been defined only partially to keep the proof clear. It is trivial to extend them to obtain complete functions preserving conditions (1), (2) and (3). ■

Now, we have to see that $\mathcal{C}_{\mathrm{FOR}(\{\mathrm{NOT},\mathrm{AND}\})}$ is pwm-reducible to $\mathcal{C}_{\mathrm{FOR}(F)}$ if F is able to "generate" any of the following functions: $\varphi_{F_2^\infty}$, $\varphi_{F_6^\infty}$ or φ_{D_2}. Let us do a case analysis.

Theorem 2 *Let F be a set of boolean functions. If $\varphi_{F_2^\infty} \in \langle F \rangle$ then $\mathcal{C}_{\mathrm{FOR}(\{\mathrm{AND},\mathrm{OR}\})}$ is pwm-reducible to $\mathcal{C}_{\mathrm{FOR}(F)}$.*

Proof. Clone $\langle F \rangle$ includes the operation $\varphi_{F_2^\infty}(x, y, z) = x \vee (y \wedge z)$. Thus, there exists some formula $\Phi_{F_2^\infty}$ in $\mathrm{FOR}(F)$ over three variables x, y, z such that $[\Phi_{F_2^\infty}] = \varphi_{F_2^\infty}$.

Clearly, with the operation $\varphi_{F_2^\infty}$ and the additional help of constant 0 it is possible to simulate functions AND and OR,

$$\mathrm{AND}(x, y) = \varphi_{F_2^\infty}(0, x, y)$$

$$\mathrm{OR}(x, y) = \varphi_{F_2^\infty}(x, y, y)$$

Let Ψ be an arbitrary monotone boolean formula in $\mathrm{FOR}(\{\mathrm{AND}, \mathrm{OR}\})$ over the input variables $x_1, x_2 \ldots, x_n$.

Let Υ_2 be the boolean formula over the input variables $x_1, \ldots, x_n, c_0$ obtained from Ψ by replacing every occurrence $\mathrm{AND}(x, y)$ by $\Phi_{F_2^\infty}(c_0, x, y)$ and, similarly, every occurrence of $\mathrm{OR}(x, y)$ by $\Phi_{F_2^\infty}(x, y, y)$.

Finally, let Υ_1 be the boolean formula defined by:

$$\Upsilon_1(x_1, x_2, \ldots, x_n, c_0) = \Phi_{\Upsilon_2^\infty}(\Upsilon_2(x_1, x_2 \ldots, x_n, c_0), c_0, c_0),$$

By construction we have

$$\Upsilon_2(x_1, x_2, \ldots, x_n, 0) = \Psi(x_1, x_2, \ldots, x_n), \text{and}$$

$$\Upsilon_1(x_1, x_2, \ldots, x_n, c_0) = \begin{cases} \Psi(x_1, x_2, \ldots, x_n) & \text{if } c_0 = 0 \\ 1 & \text{otherwise} \end{cases}$$

Now we are in position to define the pwm-reduction. Let g be the function assigning to every monotone boolean formula Ψ, an associated formula Υ_1 in FOR(F) constructed as described above. Let f be the function adding the value of the constant zero to the end of string, i.e.,

$$f(s, n, \langle x_1, x_2, \ldots, x_n \rangle) = \langle x_1, x_2, \ldots, x_n, 0 \rangle.$$

Mapping h produces the inverse result. That is, given an string removes the last value (corresponding to the constant 0).

$$h(s, n, \langle x_1, \ldots, x_n, c_0 \rangle) = \langle x_1, \ldots, x_n \rangle$$

Finally, function j is defined by

$$j(s, n, \langle x_1, \ldots, x_n, c_0 \rangle, b) = \begin{cases} b & \text{if } c_0 = 0 \\ \top & \text{otherwise} \end{cases}$$

Thus, it is immediate to verify that f, g, h, and j define a pwm-reduction from $\mathcal{C}_{\{\text{AND},\text{OR}\}}$ to $\mathcal{C}_{\text{FOR}(F)}$. ∎

By duality we have,

Theorem 3 *Let F be a set of boolean functions. If $\varphi_{F_6}^{\infty} \in \langle F \rangle$ then $\mathcal{C}_{\text{FOR}(\{\text{AND},\text{OR}\})}$ is pwm-reducible to $\mathcal{C}_{\text{FOR}(F)}$.*

Finally we study clones containing φ_{D_2}.

Theorem 4 *Let F be a set of boolean functions. If $\varphi_{D_2} \in \langle F \rangle$ then $\mathcal{C}_{\text{FOR}(\{\text{AND},\text{OR}\})}$ is pwm-reducible to $\mathcal{C}_{\text{FOR}(F)}$.*

Proof. For this proof we will use the fact that operation φ_{D_2} satisfies the self-duality property. Thus, every function in $\langle \varphi_{D_2} \rangle$ is self-dual. Clone $\langle F \rangle$ includes the majority operation $\varphi_{D_2}(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$. Thus, there exists some formula Φ_{D_2} in FOR(F) over three variables x, y, z such that the function computed by F_{D_2} is φ_{D_2}.

Clearly, with the operation φ_{D_2} and the additional help of constants it is possible to simulate functions AND and OR,

$$\text{AND}(x, y) = \varphi_{D_2}(0, x, y)$$

$$\mathrm{OR}(x,y) = \varphi_{D_2}(1,x,y)$$

For every monotone boolean formula Ψ in $\mathrm{FOR}(\{\mathrm{AND},\mathrm{OR}\})$ over the input variables $x_1, x_2 \ldots, x_n$ we construct an associated formula Υ_1 over the variables $x_1, x_2 \ldots, x_n, c_0, c_1$ defined by

$$\Upsilon_1(x_1, x_2, \ldots, x_n, c_0, c_1) = \Phi_{D_2}(\Upsilon_2(x_1, x_2 \ldots, x_n, c_0, c_1), c_0, c_1),$$

where Υ_2 is the formula over the input variables $x_1, x_2, \ldots, x_n, c_0, c_1$ obtained from Ψ in a similar way to the previous proof: we replace every occurrence of $\mathrm{AND}(x,y)$ by $\Phi_{D_2}(c_0,x,y)$ and every occurrence of $\mathrm{OR}(x,y)$ by $\Phi_{D_2}(c_1,x,y)$.

By construction we have (the case $c_0 = 1 \wedge c_1 = 0$ is a consequence of the self-duality of D_2),

$$\Upsilon_2(x_1, x_2, \ldots, x_n, c_0, c_1) = \begin{cases} \Psi(x_1, x_2, \ldots, x_n) & \text{if } c_0 = 0 \wedge c_1 = 1 \\ \neg\Psi(\neg x_1, \neg x_2, \ldots, \neg x_n) & \text{if } c_0 = 1 \wedge c_1 = 0 \\ \text{undetermined} & \text{otherwise} \end{cases}$$

$$\Upsilon_1(x_1, x_2, \ldots, x_n, c_0, c_1) = \begin{cases} \bot & \text{if } c_0 = c_1 = 0 \\ \top & \text{if } c_0 = c_1 = 1 \\ \Psi(x_1, x_2, \ldots, x_n) & \text{if } c_0 = 0 \wedge c_1 = 1 \\ \neg\Psi(\neg x_1, \neg x_2, \ldots, \neg x_n) & \text{if } c_0 = 1 \wedge c_1 = 0 \end{cases}$$

Now we are in position to define the pwm-reduction. Let g be the function assigning to every monotone boolean formula Ψ in $\mathrm{FOR}(\{\mathrm{AND},\mathrm{OR}\})$, an associated formula Υ_1 in $\mathrm{FOR}(F)$ constructed as described above. Let f be the function adding the value of the constants to the end of string, i.e.,

$$f(s, n, \langle x_1, x_2, \ldots, x_n\rangle) = \langle x_1, x_2, \ldots, x_n, 0, 1\rangle.$$

Mapping h removes the last two values (corresponding to the constants) and, moreover, h negates the values of the assignment if the values of the constants are flipped,

$$h(s, n, \langle x_1, \ldots, x_n, c_0, c_1\rangle) = \begin{cases} \langle x_1, \ldots, x_n\rangle & \text{if } c_0 = 0 \vee c_1 = 1 \\ \langle \neg x_1, \ldots, \neg x_n\rangle & \text{otherwise} \end{cases}$$

Finally, function j is defined by

$$j(s, n, \langle x_1, \ldots, x_n, c_0, c_1\rangle, b) = \begin{cases} 0 & \text{if } c_0 = 0 \wedge c_1 = 0 \\ b & \text{if } c_0 = 0 \wedge c_1 = 1 \\ \neg b & \text{if } c_0 = 1 \wedge c_1 = 0 \\ 1 & \text{if } c_0 = 1 \wedge c_1 = 1 \end{cases}$$

Thus, it is immediate to verify that f, g, h, and j define a pwm-reduction from $\mathcal{C}_{\{\mathrm{AND},\mathrm{OR}\}}$ to $\mathcal{C}_{\mathrm{CIR}(F)}$. ■

References

[1] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.

[2] D. Angluin, M. Frazier, and L. Pitt. Learning Conjunctions of Horn Clauses. *Machine Learning*, 9:147–164, 1992.

[3] D. Angluin, L. Hellerstein, and M. Karpinski. Learning Read-Once Formulas with Queries. *Journal of the ACM*, 40:185–210, 1993.

[4] D. Angluin and M. Kharitonov. When won't Membership Queries help. *Journal of Computer and System Sciences*, 50:336–355, 1995.

[5] U. Berggren. Linear Time Deterministic Learning of k-term DNF. In *6th Annual ACM Conference on Computational Learning Theory, COLT'93*, pages 37–40, 1993.

[6] N. Creignou. A Dichotomy Theorem for Maximum Generalized Satisfiability Problems. *Journal of Computer and System Sciences*, 51(3):511–522, 1995.

[7] N. Creignou and M. Hermann. Complexity of Generalized Satisfiability Counting Problems. *Information and Computation*, 125:1–12, 1996.

[8] V. Dalmau. A Dichotomy Theorem for Learning Quantified Boolean Formulas. *Machine Learning*, 35(3):207–224, 1999.

[9] V. Dalmau and P. Jeavons. Learnability of Quantified Formulas. In *4th European Conference on Computational Learning Theory Eurocolt'99*, volume 1572 of *Lecture Notes in Artificial Intelligence*, pages 63–78, Berlin/New York, 1999. Springer-Verlag.

[10] L.M. Goldschlager. A Characterization of Sets of n-Input Gates in Terms of their Computational Power. Technical Report 216, Basser Department of Computer Science, The University of Sidney, 1983.

[11] L.M. Goldschlager and I. Parberry. On the Construction of Parallel Computers from various bases of Boolean Circuits. *Theoretical Computer Science*, 43:43–58, 1986.

[12] P. Jeavons, D. Cohen, and M.C. Cooper. Constraints, Consistency and Closure. *Artificial Intelligence*, 101:251–265, 1988.

[13] P. Jeavons, D. Cohen, and M. Gyssens. A Unifying Framework for Tractable Constraints. In *1st International Conference on Principles and Practice of Constraint Programming, CP'95, Cassis (France), September 1995*, volume 976 of *Lecture Notes in Computer Science*, pages 276–291. Springer-Verlag, 1995.

[14] P. Jeavons, D. Cohen, and M. Gyssens. A Test for Tractability. In *2nd International Conference on Principles and Practice of Constraint Programming CP'96*, volume 1118 of *Lecture Notes in Computer Science*, pages 267–281, Berlin/New York, August 1996. Springer-Verlag.

[15] P. Jeavons, D. Cohen, and M. Gyssens. Closure Properties of Constraints. *Journal of the ACM*, 44(4):527–548, July 1997.

[16] P. Jeavons and M. Cooper. Tractable Constraints on Ordered Domains. *Artificial Intelligence*, 79:327–339, 1996.

[17] D. Kavvadias and M. Sideri. The Inverse Satisfiability Problem. In *2nd Computing and Combinatorics COCOON'96*, volume 1090 of *Lecture Notes in Computer Science*, pages 250–259. Springer-Verlag, 1996.

[18] Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM*, 41(1):67–95, January 1994.

[19] S. Khanna, M. Sudan, and L. Trevisan. Constraint Satisfaction: The Approximability of Minimization Problems. In *12th IEEE Conference on Computational Complexity*, 1997.

[20] S. Khanna, M. Sudan, and P. Williamson. A Complete Classification fo the Approximability of Maximation Problems Derived from Boolean Constraint Satisfaction. In *29th Annual ACM Symposium on Theory of Computing*, 1997.

[21] E.L. Post. *The Two-Valued Iterative Systems of Mathematical Logic*, volume 5 of *Annals of Mathematics Studies.* Princeton, N.J, 1941.

[22] S. Reith and H. Vollmer. The Complexity of Computing Optimal Assignments of Generalized Propositional Formulae. Technical Report TR196, Department of Computer Science, Universität Würzburg, 1999.

[23] S. Reith and K.W. Wagner. The Complexity of Problems Defined by Subclasses of Boolean Functions. Technical Report TR218, Department of Computer Science, Universität Würzburg, 1999.

[24] T.J. Schaefer. The Complexity of Satisfiability Problems. In *10th Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.