Exact learning Via the Monotone Theory

Nader H. Bshouty
Summed by: Erez Tsidon
Department of Computer Science
Technion 32000
Israel

Abstract

Any boolean function is learnable in polynomial time in its minimal CNF and DNF size and the number of variables n. In particular, Decision trees are learnable in polynomial time.

1 INTRODUCTION

In this lecture note we develop new techniques for exact learning boolean functions.We show:

1. Any boolean function is learnable in polynomial time in its minimal DNF size, its minimal CNF size, and the number of variables n.
2. Decision trees are learnable in polynomial time.

Our algorithms are in the model of exact learning with membership queries and unrestricted equivalence queries. The hypothesis to the equivalence queries and the output hypothesis are depth 3 formulas.

Denote $Size_{DNF}(f)$ and $Size_{CNF}(f)$ the minimal number of terms and clauses over all possible DNF and CNF formulas of f, respectively. We denote by $Size_{DT}(f)$ the minimal number of leaves over all possible Decision tree representations of f.

Lemma 1 *$DT_n \subseteq DNF_n$ and $DT_n \subseteq CNF_n$.*

This means that any polynomial size Decision Tree is polynomial size DNF and CNF.

Proof. Denote by f_{DT} a Decision tree representation of f. Each leaf u labeled with 1 in f_{DT} corresponds to a term T_u such that: $T_u(x_0) = 1$ if and only if the computation of $f_{DT}(x_0)$ ends at leaf u. If $v_1 \rightarrow^{\xi_1} v_2 \rightarrow^{\xi_2} \cdots \rightarrow^{\xi_{r-1}} v_r \rightarrow^{\xi_r} u$ is the path to u and $\xi_1, ..., \xi_r$ are the labels of the edges then $T_u(x) = x_{i_1}^{\xi_1} \cdots x_{i_r}^{\xi_r}$ where $x^\xi = x$ if $\xi = 1$ and $x^\xi = \bar{x}$ otherwise and $x_{i_1}, ..., x_{i_r}$ are the labels of $v_1, ..., v_r$. f_{DT} can be written now as DNF,

$$f_{DT} = \bigvee_{u \text{ is 1's leaf}} T_u$$

Producing CNF is dual by taking the pathes from the root to $0's$ leaves ,writing them as clauses and making a conjunction of all of those clauses, so $DT \subseteq CNF$.□

Denote $Size_{DT}(f)$ the number of leaves of f_{DT}. We now prove that $Size_{DT}(f)$ is greater or equal to the sum of $Size_{DNF}(f)$ and $Size_{CNF}(f)$. This means that if f is learnable in $Poly(n, Size_{DNF}, Size_{CNF})$ then f_{DT} is learnable in $Poly(n, Size_{DT})$.

Lemma 2 $Size_{DT}(f) \geq Size_{DNF}(f) + Size_{CNF}(f)$

Proof. The previous Lemma implies that the number of the leafs equals to the number of terms in some DNF representation of f plus the number of clauses in some CNF representation of f. Therefore the number of leafs is greater or equal to the number of terms of the minimal size DNF representation of f plus the number of clauses of the minimal size CNF representation of f.□

2 THE LEARNING MODEL

In the Exact learning model there is a function f, called target function, which is a member of a class of functions C defined over the variable set $V = \{x_1, ..., x_n\}$. The goal of the learner is to halt and output a formula h that is logically equivalent to f. The teacher can answer two types of queries, Equivalence Queries(EQ) and Membership Queries(MQ). The learner sends a function h to the teacher as EQ from a class of functions H. The reply of the oracle is either "yes", signifying that h is equivalent to f, or a counterexample, which is an assignment a such that $h(a) \neq f(a)$. In the Exact learning model the learner asks only EQs. In the $Exact(MQ)$ learning model the learner have also the ability to ask Membership Queries. In the MQ the learner sends to the teacher an assignment a and the teacher returns $f(a)$.

The PAC(MQ) model is a PAC learning model with the ability of asking MQ's. PAC_U model is a PAC learning model where the examples $(a, f(a))$ that the teacher gives to the learner are uniformly distributed over all possible examples $(a, f(a))$. The PAC_U(MQ) model is just the same as PAC_U where the learner have also the ability to ask MQ's.

We prove our theorem over the Exact(MQ) learning model but it is also true for the PAC(MQ) and PAC_U(MQ) learning models.

2.1 LEARNING MODELS RELATIONSHIPS

$$\begin{array}{ccc}
EXACT & \Rightarrow & EXACT(MQ) \\
\Downarrow & & \Downarrow \\
PAC & \Rightarrow & PAC(MQ) \\
\Downarrow & & \Downarrow \\
PAC_U & \Rightarrow & PAC_U(MQ)
\end{array}$$

In this lecture we prove the following Theorem:

Theorem 2.1.1 *Any function, $f(x_1,\ldots,x_n)$, is learnable in $Poly(n, Size_{DNF}(f), Size_{CNF}(f))$ time. In particular, Decision trees are learnable in polynomial time.*

This is the main Theorem of this lecture note.

3 THE MONOTONE THEORY

Definition 1 *Monotone Term(MTERM) is a term that includes only positive variables. Monotone DNF(MDNF) is a DNF that includes only MTERMS. Monotone function is any boolean function that can be represented with positive variables and the $\vee$ and $\wedge$ gates.*

Definition 2 *Define the order "$\geq$" between two assignments, a and b, by:*

$$a \geq b \Leftrightarrow \forall i : a_i \geq b_i$$

Denote by $wt(a)$ the Hamming weight of the assignment a. We define the following lattice on $\{0,1\}^n$. Each assignment $a \in \{0,1\}^n$ is a vertex in a direct acycles graph. Assignment a is connected to assignment b if and only if $wt(a) = wt(b) + 1$ and $a \geq b$.

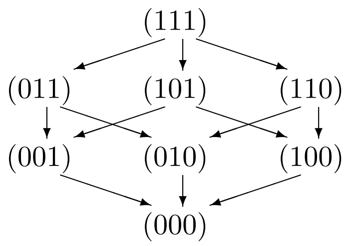


Figure 2.1: Example for n=3 lattice

A walk down tour in the lattice $\{0,1\}^n$ is $a_1 \to a_2 \to \cdots \to a_k$ where each a_i is a_{i-1} with one bit flipped from 1 to 0. This tour create a path in the lattice.

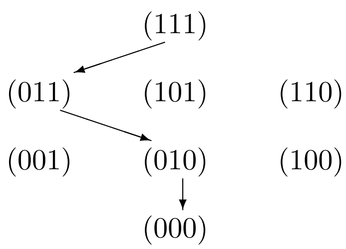


Figure 2.2: A walk down tour from (111) to (000)

Lemma 3 *We have: the function f is Monotone function if and only if for every $x \leq y$ we have $f(x) \leq f(y)$.*

Proof:

- ($\Rightarrow$) Let f be a Monotone function we will prove the first direction by induction over the number of gates in f. For $f \equiv x_0$ it is obvious that $x \leq y \Rightarrow f(x) \leq f(y)$. Now, let f and g be Monotone functions. Proving the $i+1$ iteration is by checking wether the following functions are Monotone: $f \vee g$ and $f \wedge g$. We have: $x \leq y \Rightarrow f(x) \leq f(y)$ and $g(x) \leq g(y)$. The options for $f \vee g$ and $f \wedge g$ are:

$f(x)$	$f(y)$	$g(x)$	$g(y)$	$f(x) \vee g(x)$	$f(y) \vee g(y)$	$f(x) \wedge g(x)$	$f(y) \wedge g(y)$
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	1	1	1	0	1
1	1	0	0	1	1	0	0
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

  This means that $x \leq y \Rightarrow f(x) \vee g(x) \leq f(y) \vee g(y)$ and $f(x) \wedge g(x) \leq f(y) \wedge g(y)$. This proves the $i+1$ iteration.

- ($\Leftarrow$) Let $f = T \vee g$ be monotone and suppose T contains some negated variables. Consider the monotone term M that contains all the positive variables in T. Then $M \vee T = M$. Consider the minimal assignment a that satisfies T. Then,

$$T(a) = M(a) = f(a) = 1$$

  Since for all $b > a$ we also have $M(b) = f(b) = 1$ and since $M(c) = 1$ implies $c > a$ we have $M \Rightarrow f$ and

$$f = M \vee f = M \vee T \vee g = M \vee g.\square$$

This Lemma implies that in the lattice, for any MTERM there is some assignment, a, such that $f = 1$ for a and for every point larger than a, and *f=0* otherwise. we call this assignment a Minterm.

Definition 3 *The Minterm of MTERM, T, is an assignment, a(T), such that:*

$$T(b) = \begin{cases} 1 & b \geq a(T) \\ 0 & \textit{otherwise} \end{cases}$$

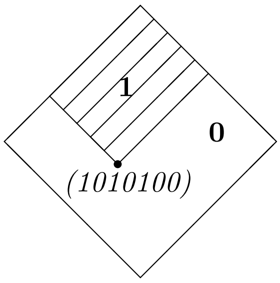


Figure 2.3: The lattice of the MTERM - $x_1x_3x_5$ (n=7).
a(T) = (1010100)

In the same way the lattice of a MDNF has several Minterms, one for each MTERM.

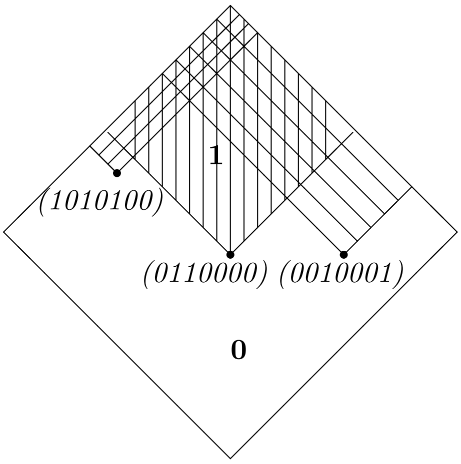


Figure 2.4: The lattice of the MDNF - $x_1x_3x_5 \vee x_2x_3 \vee x_3x_7$

Before we continue we define:

- For assignment a, $T_a = \bigwedge_{a_i=1} x_i$
- b is son of a if b is the result of one step walk down tour in the lattice, starting from a.

Now we can describe a simple algorithm for learning MDNF. First we show how to learn a MTERM using only membership queries. As it shown above, for learning a MTERM we need to find its minterm. Testing whether an assignment a is a minterm is done by checking if $f(a) = 1$ and then checking whether $\forall i : f(b_i) = 0$ where $\{b_i\}_{i=0}^{m}$ is the set of a sons (m is the number of $1's$ bits in a, and then the number of its sons). If this true a is a minterm, otherwise we take one of a sons, b_{i_0}, such that $f(b_{i_0}) = 1$ and do the same process we did to a. We continue to walk down in the lattice until we get to a minterm.

The algorithm for learning MDNF f at stage r finds r minterms $a^{(i_1)}, ..., a^{(i_r)}$. At stage $r+1$ the algorithm asks equivalence query $EQ(h)$ where $h = T_{a^{(i_1)}} \vee \cdots \vee T_{a^{(i_r)}}$ (the first equivalence query asked by the algorithm is $EQ(0)$). As we will prove later, $h \Rightarrow f$ therefore the only counterexample a possible is one that satisfies $f(a) = 1$ and $h(a) = 0$. Now we run the previous algorithm to find a minterm starting from a. Notice that since $h(a) = 0$ and

h is monotone, the assignments that we get as a result of the walk down tour from a are still counterexamples for f. If a is the new minterm we found then since $h(a) = 0$ we have $T_{a^{(i_j)}}(a) = 0$ for all j and therefore $a \neq a^{(i_j)}$ for all j and a new minterm has been found. We keep running the algorithm until all minterms are found. This algorithm is known as the Angluin Algorithm.

Angluin Algorithm:
$h \leftarrow 0$
While $EQ(h) \neq$ "YES" do
  Let a be a counterexample.
  While $\exists b$: b son of a and $f(b) = 1$ do
    $a \leftarrow b$
  $h \leftarrow h \vee T_a$
Output(h)

Complexity. Every walk down tour has at most n levels (lattice height), each assignment has at most n sons (number of 1 bits), therefore we have at most n^2 MQs for each tour. For each MTERM we make a tour and ask an EQ so we have $Size_{MDNF}(f)$ EQs. Therefore, the total time complexity is $n^2 Size_{MDNF}(f)$.

IMPORTANT PROPERTIES OF AA:

1. The hypothesis h of the EQ satisfies $h \Rightarrow f$.

   Proof. Induction over iterations: In the first iteration $h \equiv 0 \Rightarrow f$. In the $i + 1$ iteration we find a new MTERM of f_{MDNF}, T_a. Since $T_a \Rightarrow f$ we have $h_{i+1} = h_i \vee T(a) \Rightarrow f$. □

2. All counterexamples are positive, i.e. $f(a) = 1$. (This follows from 1.)

We now ask the following question:

What happens if we run AA on any TERM? or any DNF?

Let's take the term: $x_1 x_2 \overline{x_5} x_6 \overline{x_7}$ (n=8) and run Angluin algorithm for this term. In the first step the algorithm asks $EQ(0)$. Let's say the oracle returns $b = (11110100)$ as a counterexample. Flipping the third and the fourth bits in b will produce a minterm $a = (11000100)$ that correspond to the MTERM $T_a = x_1 x_2 x_6$. But now, for $EQ(x_1 x_2 x_6)$ we can get a negative counterexample, that is a counterexample a such that $h(a) = 1$ and $f(a) = 0$, for example - (11000101).

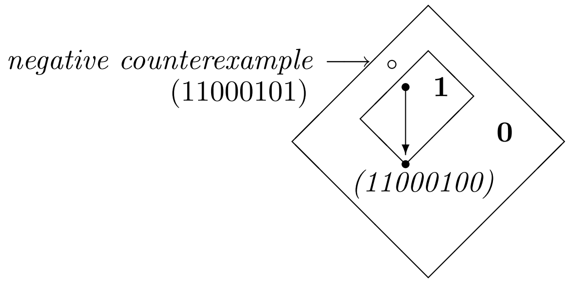


Figure 2.5: Result of AA on - $T = x_1 x_2 \overline{x_5} x_6 \overline{x_7}$

Before we continue let us define the minimal monotone function that contains f:

Definition 4 *The Minimal Monotone Function $\mathcal{M}(f)$ of f is defined as follows:*

$$\mathcal{M}(f)(x) = \begin{cases} 1 & \exists y \leq x : f(y) = 1 \\ 0 & \textit{otherwise} \end{cases}$$

Now, suppose we have an oracle PEQ that returns only positive counterexamples. Running AA with PEQ on DNF f will produce a monotone function h. We will show that h is actually $\mathcal{M}(f)$.

Lemma 4 *The output, h, of Angluin algorithm with PEQ on DNF f is $\mathcal{M}(f)$.*

Proof: Angluin algorithm will stop when there are no more positive counterexamples to h. That is, when $f \Rightarrow h$. Therefore $h(a) = 0 \Rightarrow f(a) = 0$ and because h is monotone $\forall b \leq a : h(b) = 0$ that means $\forall b \leq a : f(b) = 0$ so by definition $\mathcal{M}(f)(a) = 0$ and therefore $\mathcal{M}(f) \Rightarrow h$. Now, if $\mathcal{M}(f)(a) = 0$ it implies that $\forall b \leq a : f(b) = 0$. This means that AA couldn't make a walk down tour to find a minterm lower than a. Further more, a itself can't be the minterm that AA found because $f(a) = 0$. Therefore it must be that $h(a) = 0$. This means $h \Rightarrow \mathcal{M}(f)$. So $h = \mathcal{M}(f)$.$\square$

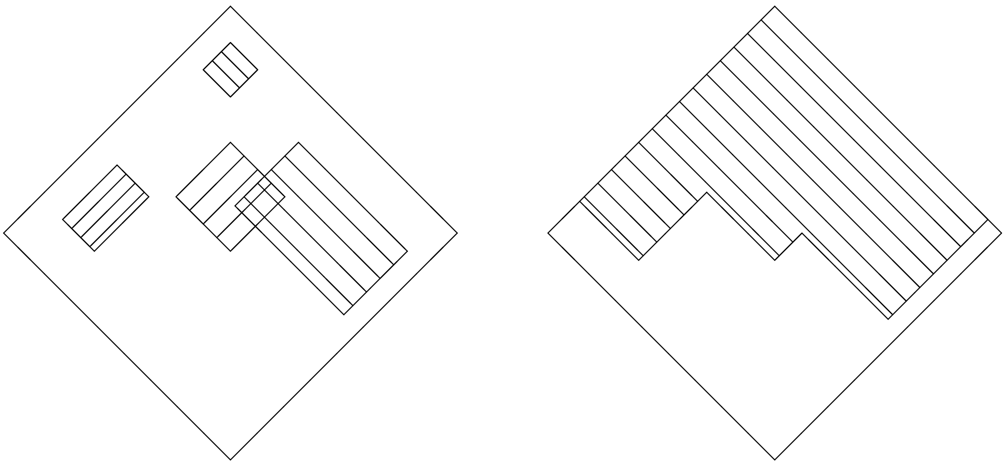


Figure 2.6: The lattice of the DNF - $\overline{x_1 x_2} x_5 \vee x_1 \overline{x_2} x_3 x_4 x_5 \vee x_1 x_3 \overline{x_5} \vee \overline{x_2} x_3 x_4$
and the AA result: $\mathcal{M}(f) = x_5 \vee x_1 x_3 x_4 x_5 \vee x_1 x_3 \vee x_3 x_4$

We will show that the complexity of AA running on DNF is remaining $n^2Size_{DNF}(f)$, just like in the MDNF case.

Lemma 5 *Let f be a DNF with r terms. The number of $PEQ's$ asked by AA while running on f is at most r.*

Proof: We prove the lemma by induction over the number of terms in f. Let $f = T$ be a term, and let a be the first minterm that AA found. Notice that $f(a) = 1$ and $\forall b < a : f(b) = 0$, so for every $a_i = 1$ x_i must be in T, otherwise we could flip a_i to 0 and get a lower assignment b such that $f(b) = 1$. This means $f \Rightarrow T_a$ so $PEQ(T_a)$ will be answered by "YES", and AA finished after one PEQ. Now let's look at $f = T_1 \vee ... \vee T_i \vee T_{i+1}$. After $j \leq i$ $PEQ's$ asked by AA, $g = T_1 \vee ... \vee T_i \Rightarrow h = T_{a^1} \vee ... \vee T_{a^j}$, where $a^1, ..., a^j$ are the j minterms AA found. If $T_{i+1} \Rightarrow h$ then $f = g \vee T_{i+1} \Rightarrow h$ and AA finished after $j \leq i$ $PEQ's$. If not, the next minterm that AA will find, c, will be such that $T_{i+1} \Rightarrow T_c$ (The same explanation like the first case) so $f = g \vee T_{i+1} \Rightarrow h \vee T_c$ and we finish after $j + 1 \leq i + 1$ $PEQ's$. $\Box$

Notice that the number of $MQ's$ needed to find each minterm is remaining n^2. So we have n^2 $MQ's$ for each PEQ, and we have at most $Size(DNF)$ $PEQ's$. Therefore the total time complexity is $n^2Size(DNF)$.

Before we continue we will prove some results about $\mathcal{M}(f)$:

Lemma 6 $\mathcal{M}(x_{i_1} \cdots x_{i_k} \cdot \overline{x_{j_1}} \cdots \overline{x_{j_l}}) = x_{i_1} \cdots x_{i_k}$

Proof: We have

$$\mathcal{M}(x_{i_1} \cdots x_{i_k} \cdot \overline{x_{j_1}} \cdots \overline{x_{j_l}})(a) = 1$$

$$\Leftrightarrow \exists b \leq a : b_{i_1} \cdots b_{i_k} \cdot \overline{b_{j_1}} \cdots \overline{b_{j_l}} = 1$$

$$\Leftrightarrow \exists b \leq a : (b_{i_1}, ..., b_{i_k}, b_{j_1}, ..., b_{j_l}) = (1, ..., 1, 0, ..., 0)$$

$$\Leftrightarrow (a_{i_1}, ..., a_{i_k}) = (1, ..., 1)$$

$$\Leftrightarrow x_{i_1} \cdots x_{i_k}(a) = 1 \ \Box$$

Lemma 7 $\mathcal{M}(f \vee g) = \mathcal{M}(f) \vee \mathcal{M}(g)$

Proof: We have

$$\mathcal{M}(f \vee g)(a) = 1$$

$$\Leftrightarrow (\exists b \leq a)(f(b) = 1 \vee g(b) = 1)$$

$$\Leftrightarrow (\exists b \leq a)f(b) = 1 \vee (\exists b \leq a)g(b) = 1$$

$$\Leftrightarrow \mathcal{M}(f)(a) = 1 \vee \mathcal{M}(g)(a) = 1$$

$$\mathcal{M}(f)(a) \vee \mathcal{M}(g)(a) = 1 \ \Box$$

Lemma 8 $\mathcal{M}(f \wedge g) \Rightarrow \mathcal{M}(f) \wedge \mathcal{M}(g)$

Proof: We have

$$\mathcal{M}(f \wedge g)(a) = 1$$

$$\Leftrightarrow (\exists b \leq a)(f(b) = 1 \wedge g(b) = 1)$$

$$\Rightarrow (\exists b \leq a)f(b) = 1 \wedge (\exists b \leq a)g(b) = 1$$

$$\Leftrightarrow \mathcal{M}(f)(a) = 1 \wedge \mathcal{M}(g)(a) = 1$$

$$\mathcal{M}(f)(a) \wedge \mathcal{M}(g)(a) = 1 \ \Box$$

Notice that the definition of monotone functions depends on the order $\leq$. We may now change this order so learning f by AA will give us a different $\mathcal{M}(f)$. For example, let's take $f = T$ where T is a term. We can learn f using the original order and then learn it using the opposite order by replacing the $\leq$ order to $\geq$. The visual meaning of this in the lattice is making a walk up tour instead of walk down tour. Now we can find the intersection of the two $\mathcal{M}(f)$ that we found and get T. This method will not work for more than one term as it shown in the figure below.

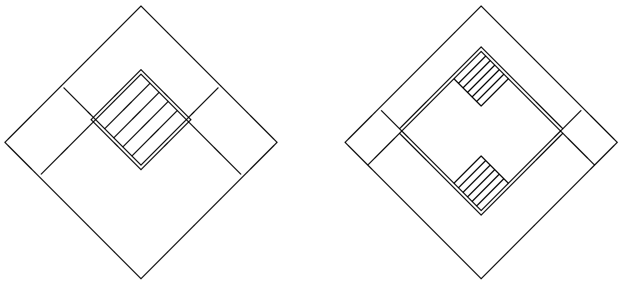


Figure 2.7: One term learning $Vs.$ two terms learning.

We can now generalize the order definition so we get a variety of tour directions in the lattice.

Definition 5 *An order "$\geq_a$" between two assignments, x and y, is defined by:*

$$x \geq_a y \Leftrightarrow x \oplus a \geq y \oplus a$$

Where $\oplus$ is a bitwise xor.

We can define now a new lattice. We call it a-lattice. The a-lattice is just the same as the original lattice beside that each assignment b in the original lattice become $b \oplus a$ in the a-lattice. Notice that for any assignment b in the a-lattice, if c is son of b then $b \geq_a c$.

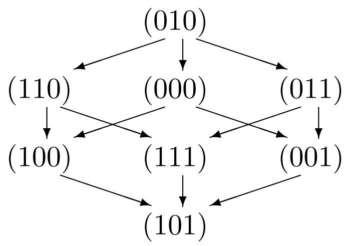


Figure 2.8: Example of a-lattice, a=(1,0,1)

We denote by $\mathbf{1}$ the assignment $(1,1,...,1)_n$ and by $\mathbf{0}$ the assignment $(0,0,...,0)_n$. Now, notice that:

1. The smallest assignment in a-lattice is a itself, and the biggest is $a+\mathbf{1}$.
2. Instead of flipping 1's bits to 0's to get the sons of some assignment b, we flip now the bits of b where $b_i \neq a_i$ from b_i to a_i.

According to Definition 5 we can change some previous Definitions and Theorems:

- *f is called a-Monotone Function if and only if f(x+a) is Monotone Function*
- *The minimal a-monotone function $\mathcal{M}_a(f)$ of f is defined as follows:*

$$\mathcal{M}_a(f)(x) = \begin{cases} 1 & \exists y \leq_a x : f(y) = 1 \\ 0 & \textit{otherwise} \end{cases}$$

- Instead of using $T_b = \bigwedge_{b_i=1} x_i$ we will use from now on $T_b^a = \bigwedge_{b_i \neq a_i} x_i^{b_i}$ where $x^1 \equiv x$ and $x^0 \equiv \bar{x}$.
- Lemma 7 and Lemma 8 are true also for $\mathcal{M}_a(f)$ (just by changing the "$\leq$" signs to "$\leq_a$" signs).
- Lemma 6 is now being changed a little, instead of throwing the negative variables from f to get $\mathcal{M}(f)$ we now throw variables according to a. For example,

$$\mathcal{M}_{\underbrace{(1011011)}_{throwing - \{x_1,\overline{x_2},x_3,x_4,\overline{x_5},x_6,x_7\}}}(x_1x_2\overline{x_5} \vee x_1x_2\overline{x_3}x_4 \vee \overline{x_2}x_5x_7 \vee x_1\overline{x_2}x_5) = x_2 \vee x_2\overline{x_3} \vee x_5x_7 \vee x_5$$

We can describe now AA_a algorithm. AA_a algorithm is AA algorithm running on a-lattice.

<u>**AA**a</u>
$h \leftarrow a$
While $EQ(h) \neq$ "YES" do
  Let b be a counterexample.
  While $\exists c$: c a-son of b and $f(c) = 1$ do
    $c \leftarrow b$
  $h \leftarrow h \vee T_b^a$
Output(h)

Now, by getting only $PEQ's$, AA_a is learning the $\mathcal{M}_a(f)$ of f. The proof of this is just like the proof of lemma 4 where the "$\leq$" signs become "$\leq_a$", and $\mathcal{M}$ becomes $\mathcal{M}_a$. Additionally, the number of $PEQ's$ asked by AA_a while running on a boolean function f is at most $Size_{DNF}(f)$. Again, the proof of this is like the proof of lemma 5. We show it in the next lemma.

Lemma 9 *Let f be a DNF with r terms, and let a be some assignment. The number of $PEQ's$ asked by AA_a while running on f is at most r.*

Proof: We prove the lemma by induction over the number of terms in f. Let $f = T$ be a term, and let b be the first minterm that AA_a found. Notice that $f(b) = 1$ and $\forall c <_a b$: $f(c) = 0$, so for every $b_i \neq a_i$ $x_i^{b_i}$ is in T, otherwise we could flip the value of b_i and get one of its sons c where $f(c) = 1$. This means $f \Rightarrow T_b^a$ so $PEQ(T_b^a)$ will be answered by "YES", and AA_a finished after one PEQ. Now let's look at $f = T_1 \vee ... \vee T_i \vee T_{i+1}$. After $j \leq i$ $PEQ's$ asked by AA_a, $g = T_1 \vee ... \vee T_i \Rightarrow h = T_{b^1} \vee ... \vee T_{b^j}$, where $b^1, ..., b^j$ are the j minterms AA_a found. If $T_{i+1} \Rightarrow h$ then $f = g \vee T_{i+1} \Rightarrow h$ and AA_a finished after $j \leq i$ $PEQ's$. If not, the next minterm that AA will find, c, will be such that $T_{i+1} \Rightarrow T_c$ (The same explanation like the first case) so $f = g \vee T_{i+1} \Rightarrow h \vee T_c$ and we finish after $j + 1 \leq i + 1$ $PEQ's$. $\Box$

This means that the complexity of AA_a is just like the case of AA. We have n^2 $MQ's$ needed for each PEQ, and we have at most $Size_{DNF}(f)$ $PEQ's$. So the total time complexity is $n^2 Size_{DNF}(f)$.

Let's go back to the term learning process. We saw that for learning a term $f = T$ we can use AA_a where $a \in \{\mathbf{0}, \mathbf{1}\}$. We get two minimal a-monotone functions, $\mathcal{M}_{\mathbf{0}}(f)$ and $\mathcal{M}_{\mathbf{1}}(f)$, so their intersection is equal to T. We call $\{\mathbf{0}, \mathbf{1}\}$ a basis of f.

$$\begin{array}{ccccc} \mathcal{M}_{\mathbf{0}}(f) & & \mathcal{M}_{\mathbf{1}}(f) & & (PEQ) \\ \downarrow & & \downarrow & & \\ h_0 & \bigwedge & h_1 & = T & \end{array}$$

Lemma 10 *For any boolean function f we have $f = \bigwedge_{a\in\{0,1\}^n} \mathcal{M}_a(f)$.*

Proof: Notice that $f \Rightarrow \mathcal{M}_a(f)$ for any a and therefore $f \Rightarrow \bigwedge_{a\in\{0,1\}^n} \mathcal{M}_a(f)$. Now if $f(a) = 0$ then $\mathcal{M}_a(f)(a) = 0$ (a is the minimal element in th order $\leq_a$) and therefore $\bigwedge_{a\in\{0,1\}^n} \mathcal{M}_a(f) \Rightarrow f$.$\square$

Definition 6 *$A \subseteq \{0,1\}^n$ is called a Basis to f if $f \equiv \bigwedge_{a\in A} \mathcal{M}_a(f)$*

Assuming we know the basis A, and we use PEQ oracle, learning a boolean function f can be achieved by running AA_a for every $a \in A$ until we get $\mathcal{M}_a(f)$, and output the conjunction of them. This algorithm is shown below.

$$
\begin{array}{llllll}
A = & \{\ a_1 & a_2 & \ldots & a_m\ \} \\
Run & AA_{a_1} & AA_{a_2} & \ldots & AA_{a_m} \\
 & \downarrow & \downarrow & \ldots & \downarrow \\
 & h_1^* = \mathcal{M}_{a_1}(f) & h_2^* = \mathcal{M}_{a_2}(f) & \ldots & h_m^* = \mathcal{M}_{a_m}(f) \\
Compute & & h^* = \bigwedge_i h_i^* = f & &
\end{array}
$$

The complexity of this algorithm is $n^2 Size_{DNF}(f)$ for each AA_a, so the total time complexity is $|A|n^2 Size_{DNF}(f)$.

Now, let's try to get rid of the $PEQ's$. The reason we needed the PEQ is because we didn't want to get from the oracle negative counterexamples. So now we can run all the AA's in parallel and wait until all of them ask $PEQ(h_i)$ (h_i is the hypothesis that AA_i wants to check), as it shown below.

$$
\begin{array}{lllll}
A = & \{\ a_1 & a_2 & \ldots & a_m\ \} \\
 & AA_{a_1} & AA_{a_2} & \ldots & AA_{a_m} \\
 & \downarrow & \downarrow & \ldots & \downarrow \\
 & PEQ(h_1) & PEQ(h_2) & \ldots & PEQ(h_m)
\end{array}
$$

Notice that $\mathcal{M}_{a_i}(f)$ is the output of AA_{a_i} and it is actually the conjunction of all the hypothesis AA_{a_i} was asking from the oracle. Therefore it's obvious that $h_i \Rightarrow \mathcal{M}_{a_i}(f)$, because h_i is one of those hypothesis. This implies, $\bigwedge_{i=1}^m h_i \Rightarrow \bigwedge_{i=1}^m \mathcal{M}_{a_i}(f) = f$. Therefore, asking $EQ(\bigwedge_{i=1}^m h_i)$ will produce a positive counterexample, b, such that $\bigwedge_{i=1}^m h_i(b) = 0$ and $f(b) = 1$. That means that there exists i_0 such that $h_{i_0}(b) = 0$ and $f(b) = 1$ so we just need to find to which AA_{a_i} b is a counterexample, give it to them and continue to run those AA_{a_i} in parallel until they ask again $PEQ(h_i)$. The final output is the conjunction of all the AA_{a_i} final results. The complexity of this algorithm is $|A|$ times running AA so we get $|A|Size_{DNF}(f)$ equivalence queries and $|A|n^2 Size_{DNF}(f)$ membership queries. We call this algorithm **LearnByA**.

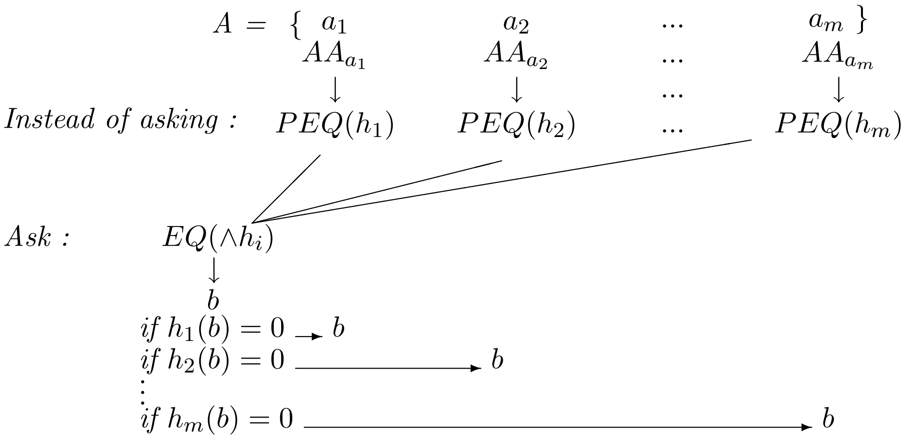


Figure 2.9 Learning f using known basis, A.

LearnByA(A):
For i=0 to $|A|$
  Run AA_{a_i} until it want to ask $PEQ(h_i)$
While $EQ(\bigwedge_{i=0}^{|A|} h_i) \neq$ "YES" do
  Let b be a counterexample
  For i=0 to $|A|$
    If $h_i(b) = 0$ then
      Return b as a counterexample to AA_{a_i}
      and continue to run it until it want to ask $EQ(h_i)$

Output($\bigwedge_{i=0}^{|A|} h_i$)

We now prove some lemmas that help us to understand how to find a basis A for some function f.

Lemma 11 $\mathcal{M}_a(f \wedge g) \Rightarrow \mathcal{M}_a(f)$

Proof: According to Lemma 8, $\mathcal{M}_a(f \wedge g) \Rightarrow \mathcal{M}_a(f) \wedge \mathcal{M}_a(g) \Rightarrow \mathcal{M}_a(f)$ $\Box$

Lemma 12 *Let C be a clause such that $C(a) = 0$, then $\mathcal{M}_a(C) = C$*

Proof: According to $\mathcal{M}_a$ definition, $\mathcal{M}_a(C)(x) = 1$ if and only if $\exists y \leq_a x : C(y) = 1$. Notice that $C(y) = 1$ if and only if $\exists i : y_i = \bar{a}_i$, and because $x \leq_a y$ this is equivalent to $\exists i : x_i = \bar{a}_i$, and this is true if and only if $C(x) = 1$. Therefore, $\mathcal{M}_a(C)(x) = 1 \Leftrightarrow C(x) = 1$, so $\mathcal{M}_a(C) = C$.$\Box$

Lemma 13 *Let A be a set of assignments, $A \subseteq \{0,1\}^n$, and let the CNF representation of some function f be: $f_{CNF} = C_1 \wedge C_2 \wedge ... \wedge C_t$. Assuming for each C_i there exists an assignment $a \in A$ such that $C_i(a) = 0$, then A is a basis to f.*

Proof: In general for any function f and for any assignment a, $f \Rightarrow \mathcal{M}_a(f)$. Therefore it's always true that $f \Rightarrow \bigwedge_{a\in A} \mathcal{M}_a(f)$. Now, let's assume that there exists $a \in A$ such that $C_i(a) = 0$, according to Lemma 11 we get $\mathcal{M}_a(f) = \mathcal{M}_a(C_1 \wedge C_2 \wedge ... \wedge C_t) \Rightarrow \mathcal{M}_a(C_i)$, and according to Lemma 12 we get $\mathcal{M}_a(f) \Rightarrow C_i$. Therefore, $\bigwedge_{a\in A} \mathcal{M}_a(f) \Rightarrow \bigwedge_{i=1}^t C_i = f$. We saw also that $f \Rightarrow \bigwedge_{a\in A} \mathcal{M}_a(f)$. So finally we get $f = \bigwedge_{a\in A} \mathcal{M}_a(f)$ □

Now we show algorithm that learns some function f without knowing its basis. The algorithm starts with $EQ(1) \rightarrow a_1$ then we define $A = \{a_1\}$ and run **LearnByA** until we get a negative counterexample, a_2. We add a_2 to A run **LearnByA** again, and so on until it stops.

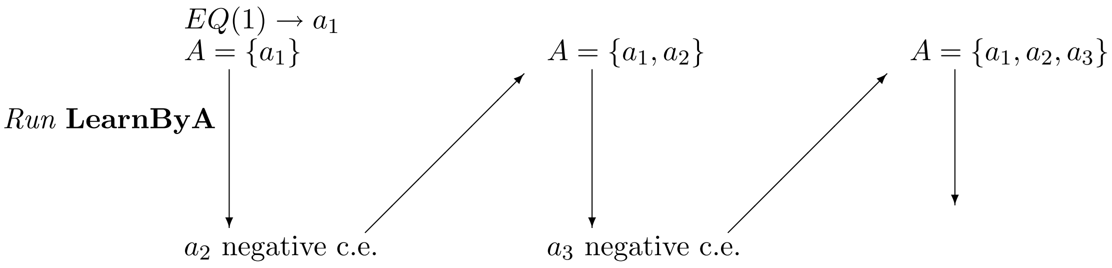


Figure 2.10 Learning f without knowing its basis.

To prove the correctness of this algorithm we should prove that every new negative counterexample is falsifying a clause of f_{CNF} that hasn't been falsified yet. So after $Size_{CNF}(f)$ steps, according to Lemma 13, we will have a full basis of f.

Lemma 14 *Let $f = f_1 \wedge f_2$ and let $A = \{a_1, ..., a_t\}$ be a set of assignments that falsify all the clauses of f_1 and not falsify even one clause of f_2. Let $h_1, ..., h_t$ be the hypothesis we build from A, so $h_1 \Rightarrow \mathcal{M}_{a_1}, ..., h_t \Rightarrow \mathcal{M}_{a_t}$. And let $b = a_{t+1}$ be a new negative counterexample, so $\bigwedge_{i=1}^t h_i(b) = 1$ and $f(b) = 0$. Then $f_2(b) = 0$.*

Result. *b, the new negative counterexample, is falsify a new clause of f.*

Proof: Because each $h_i \Rightarrow \mathcal{M}_{a_i}$ we can say $\bigwedge_{i=1}^t h_i \Rightarrow \bigwedge_{i=1}^t M_{a_i}(f)$. Now, let C_j be a clause of f_1. Notice that $M_{a_i}(f) = M_{a_i}(C_1 \wedge C_2 \wedge ...) = M_{a_i}(C_1) \wedge M_{a_i}(C_2) \wedge ... \Rightarrow M_{a_i}(C_j)$, so $\bigwedge_{i=1}^t M_{a_i}(f) \Rightarrow \bigwedge_{i=1}^t M_{a_i}(C_j)$, and therefore $\bigwedge_{i=1}^t h_i \Rightarrow \bigwedge_{i=1}^t M_{a_i}(C_j)$. Now, because A is a basis for f, there exists $a_{i_0} \in A$ such that $C_j(a_{i_0}) = 0$. Its obvious that $\bigwedge_{i=1}^t M_{a_i}(C_j) \Rightarrow M_{a_{i_0}}(C_j)$, and according to lemma 12 we get $\bigwedge_{i=1}^t M_{a_i}(C_j) \Rightarrow C_j$, so now we have

$$\bigwedge_{i=1}^{t} h_i \Rightarrow \bigwedge_{i=1}^{t} M_{a_i}(f) \Rightarrow \bigwedge_{i=1}^{t} M_{a_i}(C_j) \Rightarrow M_{a_{i_0}}(C_j) = C_j$$

This means $\bigwedge_{i=1}^t h_i(b) = 1 \Rightarrow C_j(b) = 1$. C_j is a clause of f so $\bigwedge_{i=1}^t h_i(b) = 1 \Rightarrow f_1(b) = 1$, and because $f(b) = f_1(b) \wedge f_2(b) = 0$ then it must be that $f_2(b) = 0$ □

Finally, the complete algorithm is as follows:

<u>Learnf:</u>
$N = 0$
$h_0 \equiv 1$
While $EQ(\bigwedge_{i=0}^{N} h_i) \neq$ "YES" do
  Let b be a counterexample.
  If b is a negative counterexample then
    $N \leftarrow N + 1$
    $a_N \leftarrow b$
    Run AA_{a_N} until it want to ask $EQ(h_N)$
  Else
    For i=0 to N do
      If $h_i(b) = 0$ then
        Return b as a counterexample to AA_{a_i}
        and continue to run it until it want to ask $EQ(h_i)$.

Output$(\bigwedge_{i=0}^{N} h_i)$

Complexity. We run $Size_{CNF}(f)$ different AA_a algorithms (N = basis size). For every running of AA_a we have $Size_{DNF}$ EQ and $n^2 Size_{DNF}$ MQ So we got $Size_{DNF} \cdot Size_{CNF}$ EQ and $n^2 Size_{DNF} \cdot Size_{CNF}$ MQ. Therefore, the total time complexity of the algorithm is $n^2 Size_{DNF} \cdot Size_{CNF}$. This proves Theorem 2.1.1 .

Note. The final output hypothesis is $\bigwedge_{a \in A} \mathcal{M}_a(f) \in \bigwedge DNF$, this class is called *Depth 3 formulas.*

4 LEARNING O(log(n))-termDNF

In general, the algorithm that we suggested can't learn any boolean functions in $Poly(n, Size_{DNF}(f))$ time, because the CNF size of f can be exponential big (in n). But, there are cases that the basis of the function is small, so f can be learned in $Poly(n, Size_{DNF}(f))$ time.

Example:

$$O(\log(n)) - termDNF$$

This is the class of all the boolean functions that their DNF representation has at most $k = O(log(n))$ terms,

$$T_1 \vee T_2 \vee ... \vee T_k = C_1 \wedge C_2 \wedge ...$$

Let's check the CNF size of this class. Each clause of the CNF is build from the disjunction of all the possible combinations of variables, while every variable is from different term. Therefore, the number of clauses is:

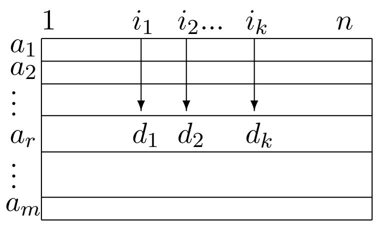
$$| T_1 | \cdot | T_2 | \cdot ... \cdot | T_k | \cong n^{O(k)}$$

and when k is not a constant the $Size_{CNF}(f)$ is not polynomial. But now we will show this class has a small basis. The CNF representation of f is from the $k - CNF$ class, that is, each clause has at most k variables (as we say, each variable is come from one of the k terms), therefore, $| C_i | \leq k$. Our goal now, is to find a set of assignments, A, where $\forall C \; \exists a \in A : C(a) = 0$. We will write each clause as follows:

$$C_i = x_{i_1}^{d_1} \vee x_{i_2}^{d_2} \vee ... x_{i_k}^{d_k}$$

$$X^d = \begin{cases} X & \text{d=0} \\ \bar{X} & \text{d=1} \end{cases}$$

C_i falsifies iff $a_{i_1} = d_1, a_{i_2} = d_2, ..., a_{i_k} = d_k$, so if we write a table:



our demand is that for every k columns and for every possible combination of $d_1, d_2, ..., d_k$ there exists some assignment, a, such that $a_{i_1} = d_1, a_{i_2} = d_2, ..., a_{i_k} = d_k$. Formally,

$$\forall 1 \leq i_1 < i_2 < ... < i_k \leq n, \forall (d_1, ..., d_k) \in \{0,1\}^k, \exists a \in A : (a_{i_1}, ..., a_{i_k}) = (d_1, ..., d_k)$$

Set of such m assignments is called *(n,k)-Universal Set.*

Let m be the number of basis elements, if $2^k log(n) \leq m \leq k 2^k log(n)$ then there exists a *(n,k)-Universal Set.* we will prove only the right inequality which implies that if $k = O(log(n))$ then $m = Poly(n)$.

Proof: we will prove it using a probability methods. We randomly choose $m = 2^k ln(\binom{n}{k}\frac{2^k}{\delta}) = O(k2^k log(n))$ assignments $a_1, ..., a_m$. Let's check the probability that they are not a *(n,k)-Universal Set*:

Pr[$\{a_1, ..., a_m\}$ *is not (n,k)-Universal Set]*$\leq$

$Pr[\exists 1 \leq i_1 < i_2 < ... < i_k \leq n, \exists(d_1, ..., d_k) \in \{0,1\}^k, \forall a : (a_{i_1}, ..., a_{i_k}) \neq (d_1, ..., d_k)]$

We have $\binom{n}{k}$ *possibilities for* $(i_1, ..., i_k)$ *and* 2^k *for* $(d_1, ..., d_k)$ *so that equals to*

$= \binom{n}{k} 2^k \prod_{i=1}^{m} Pr[(a_{i_1}, ..., a_{i_k}) \neq (d_1, ..., d_k)]$

The probability for success, $(a_{i_1}, ..., a_{i_k}) = (d_1, ..., d_k)$*, is* $\frac{1}{2^k}$ *so for failure we got*

$= \binom{n}{k} 2^k (1 - \frac{1}{2^k})^m \leq \binom{n}{k} 2^k e^{\frac{m}{2^k}} = \delta$ $\square$

So, for learning $O(log(n)) - termDNF$ in $Poly(n, Size_{DNF}(f)) = Poly(n)$ time where the probability to find a "good" basis is higher than $1 - \delta$, we should randomly choose m assignments as basis elements, while $m = Poly(n, \frac{1}{\delta})$ and run **LearnByA**.