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 \( \text{Size}_{\text{DNF}}(f) \) and \( \text{Size}_{\text{CNF}}(f) \) the minimal number of terms and clauses over all possible DNF and CNF formulas of \( f \), respectively. We denote by \( \text{Size}_{\text{DT}}(f) \) 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 \( v_1 \rightarrow^{\xi_1} v_2 \rightarrow^{\xi_2} \ldots \rightarrow^{\xi_{r-1}} v_r \rightarrow^{\xi_r} u \) is the path to \( u \) and \( \xi_1, \ldots, \xi_r \) are the labels of the edges then \( T_u(x) = x_{\xi_1}^{\xi_1} \cdots x_{\xi_r}^{\xi_r} \) where \( x^\xi = x \) if \( \xi = 1 \) and \( x^\xi = \bar{x} \) otherwise and \( x_{\xi_1}, \ldots, x_{\xi_r} \) are the labels of \( v_1, \ldots, v_r \). \( f_{DT} \) can be written now as DNF,

\[
f_{DT} = \bigvee_{u \text{ is 1's leaf}} T_u
\]
Producing CNF is dual by taking the paths from the root to 0's leaves, writing them as clauses and making a conjunction of all of those clauses, so DT ⊆ CNF. □

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

Lemma 2 $\text{Size}_{DT}(f) \geq \text{Size}_{DNF}(f) + \text{Size}_{CNF}(f)$

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

$$
\begin{align*}
\text{EXACT} & \quad \Rightarrow \quad \text{EXACT}(MQ) \\
\quad \downarrow & \quad \quad \quad \downarrow \\
\text{PAC} & \quad \Rightarrow \quad \text{PAC}(MQ) \\
\quad \downarrow & \quad \quad \quad \downarrow \\
\text{PAC}_U & \quad \Rightarrow \quad \text{PAC}_U(MQ)
\end{align*}
$$
In this lecture we prove the following Theorem:

**Theorem 2.1.1** Any function, \( f(x_1, \ldots, x_n) \), is learnable in Poly\( (n, \text{Size}_{\text{DNF}}(f), \text{Size}_{\text{CNF}}(f)) \) 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 \( \lor \) and \( \land \) gates.

**Definition 2** Define the order ”\( \geq \)” between two assignments, \( a \) and \( b \), by:

\[
a \geq b \iff \forall i : a_i \geq b_i
\]

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](image)

A walk down tour in the lattice \( \{0, 1\}^n \) is \( a_1 \rightarrow a_2 \rightarrow \cdots \rightarrow a_k \) 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)](image)
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 whether the following functions are Monotone: $f \lor g$ and $f \land g$. We have: $x \leq y \Rightarrow f(x) \leq f(y)$ and $g(x) \leq g(y)$. The options for $f \lor g$ and $f \land g$ are:

<table>
<thead>
<tr>
<th>$f(x)$</th>
<th>$f(y)$</th>
<th>$g(x)$</th>
<th>$g(y)$</th>
<th>$f(x) \lor g(x)$</th>
<th>$f(y) \lor g(y)$</th>
<th>$f(x) \land g(x)$</th>
<th>$f(y) \land g(y)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

$$f = M \lor f = M \lor T \lor g = M \lor g$$

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:

$$T(b) = \begin{cases} 
1 & b \geq a(T) \\
0 & \text{otherwise}
\end{cases}$$
In the same way the lattice of a MDNF has several Minterms, one for each MTERM.

Before we continue we define:

- For assignment $a$, $T_a = \bigwedge_{i=1}^{a_1} x_i$
- $b$ 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 $a^{(i_1)}, \ldots, a^{(i_r)}$. At stage $r+1$ the algorithm asks equivalence query $EQ(h)$ where $h = T_{a^{(i_1)}} \lor \cdots \lor T_{a^{(i_r)}}$ (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
**Angluin Algorithm:**

\[
\begin{align*}
    h &\leftarrow 0 \\
    \text{While } &\text{EQ}(h) \neq "\text{YES}" \text{ do} \\
    &\text{Let } a \text{ be a counterexample.} \\
    &\text{While } \exists b : b \text{ son of } a \text{ and } f(b) = 1 \text{ do} \\
    &\quad a \leftarrow b \\
    &\quad h \leftarrow h \lor T_a \\
    \text{Output}(h)
\end{align*}
\]

**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 \( \text{Size}_{MDNF}(f) \) EQs. Therefore, the total time complexity is \( n^2 \text{Size}_{MDNF}(f) \).

**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 \( h_{i+1} = h_i \lor T(a) \Rightarrow f \). \( \square \)

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: \( x_1 x_2 x_5 x_6 \overline{x_7} \) (\( n=8 \)) 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).
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:

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

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 $h = \mathcal{M}(f)$. □
We will show that the complexity of AA running on DNF is remaining \(n^2 \text{Size}_{\text{DNF}}(f)\), 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 \(f = T_1 \lor \ldots \lor T_i \lor T_{i+1}\). After \(j \leq i\) PEQ's asked by AA, \(g = T_1 \lor \ldots \lor T_i \Rightarrow h = T_{a_1} \lor \ldots \lor T_{a_j}\), where \(a_1, \ldots, a_j\) are the \(j\) minterms AA found. If \(T_{i+1} \Rightarrow h\) then \(f = g \lor T_{i+1} \Rightarrow h\) 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 \(f = g \lor T_{i+1} \Rightarrow h \lor T_c\) and we finish after \(j + 1 \leq i + 1\) PEQ's. \(\square\)

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 \(\text{Size}(\text{DNF})\) PEQ's. Therefore the total time complexity is \(n^2 \text{Size}(\text{DNF})\).

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

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

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

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

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

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

**Lemma 7** \(\mathcal{M}(f \lor g) = \mathcal{M}(f) \lor \mathcal{M}(g)\)

*Proof:* We have

\[
\mathcal{M}(f \lor g)(a) = 1
\]

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

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

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

\(\mathcal{M}(f)(a) \lor \mathcal{M}(g)(a) = 1\) \(\square\)
Lemma 8 \( M(f \land g) \Rightarrow M(f) \land M(g) \)

Proof: We have

\[
M(f \land g)(a) = 1 \\
\Leftrightarrow (\exists b \leq a)(f(b) = 1 \land g(b) = 1) \\
\Rightarrow (\exists b \leq a)f(b) = 1 \land (\exists b \leq a)g(b) = 1 \\
\Leftrightarrow M(f)(a) = 1 \land M(g)(a) = 1 \\
M(f)(a) \land M(g)(a) = 1
\]

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 \( M(f) \). 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 \( M(f) \) 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.](image)

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 \( b \geq_a c \).
We denote by 1 the assignment \((1,1,...,1)_n\) and by 0 the assignment \((0,0,...,0)_n\). Now, notice that:

1. The smallest assignment in a-lattice is \(a\) itself, and the biggest is \(a + 1\).
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 \(M_a(f)\) of \(f\) is defined as follows:
  \[
  M_a(f)(x) = \begin{cases} 
  1 & \exists y \leq_a x : f(y) = 1 \\
  0 & \text{otherwise}
  \end{cases}
  \]
- Instead of using \(T_b = \wedge_{b_i=1} x_i\) we will use from now on \(T^a_b = \bigwedge_{b_i \neq a_i} x^b_i\) where \(x^1 \equiv x\) and \(x^b \equiv \bar{x}\).
- Lemma 7 and Lemma 8 are true also for \(M_a(f)\) (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 \(M(f)\) we now throw variables according to \(a\). For example,
  \[
  M(1011011)(x_1x_2\bar{x}_5 \lor x_1x_2\bar{x}_3x_4 \lor \bar{x}_2x_3x_7 \lor x_1\bar{x}_2x_3) = x_2 \lor x_2\bar{x}_3 \lor x_3x_7 \lor x_5
  \]
  throwing \(-\{x_1, x_2, x_3, x_4, \bar{x}_5, x_6, x_7\}\)
We can describe now $AA_a$ algorithm. $AA_a$ algorithm is $AA$ algorithm running on a-lattice.

$$AA_a$$

\[
\begin{align*}
    h &\leftarrow a \\
    \text{While } EQ(h) \neq "YES" \text{ do} \\
    &\quad \text{Let } b \text{ be a counterexample.} \\
    &\quad \text{While } \exists c : c \text{ a-son of } b \text{ and } f(c) = 1 \text{ do} \\
    &\quad \quad c \leftarrow b \\
    &\quad h \leftarrow h \lor T_b^a \\
    \text{Output}(h)
\end{align*}
\]

Now, by getting only $PEQ'$s, $AA_a$ is learning the $M_a(f)$ of $f$. The proof of this is just like the proof of lemma 4 where the "$\leq$" signs become "$\leq_a$", and $M$ becomes $M_a$. 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 $b_i \neq a_i x_i^b$ 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 $f = T_1 \lor \ldots \lor T_i \lor T_{i+1}$. After $j \leq i$ $PEQ'$s asked by $AA_a$, $g = T_1 \lor \ldots \lor T_i \Rightarrow h = T_b^1 \lor \ldots \lor T_b^j$, where $b^1, \ldots, b^j$ are the $j$ minterms $AA_a$ found. If $T_{i+1} \Rightarrow h$ then $f = g \lor T_{i+1} \Rightarrow h$ 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 $f = g \lor T_{i+1} \Rightarrow h \lor T_c$ and we finish after $j + 1 \leq i + 1$ $PEQ'$s. □

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 $a \in \{0, 1\}$. We get two minimal a-monotone functions, $M_0(f)$ and $M_1(f)$, so their intersection is equal to $T$. We call $\{0, 1\}$ a basis of $f$.

$$
\begin{array}{ccc}
    M_0(f) & M_1(f) & (PEQ) \\
    \downarrow & \downarrow & \downarrow \\
    h_0 & h_1 & = T
\end{array}
$$
Lemma 10 For any boolean function $f$ we have $f = \bigwedge_{a \in \{0,1\}^n} M_a(f)$.

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

Definition 6 $A \subseteq \{0,1\}^n$ is called a Basis to $f$ if $f \equiv \bigwedge_a M_a(f)$.

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 $M_a(f)$, and output the conjunction of them. This algorithm is shown below.

$A = \{a_1, a_2, \ldots, a_m\}$

$\text{Run} \begin{array}{ccc}
\text{AA}_{a_1} & \text{AA}_{a_2} & \ldots & \text{AA}_{a_m} \\
\downarrow & \downarrow & \ldots & \downarrow \\
{h}^*_1 = M_{a_1}(f) & {h}^*_2 = M_{a_2}(f) & \ldots & {h}^*_m = M_{a_m}(f) \\
\text{Compute} & & & \\
{h}^* = \bigwedge_i {h}^*_i = f
\end{array}$

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

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.

$A = \{a_1, a_2, \ldots, a_m\}$

$\text{Run} \begin{array}{ccc}
\text{AA}_{a_1} & \text{AA}_{a_2} & \ldots & \text{AA}_{a_m} \\
\downarrow & \downarrow & \ldots & \downarrow \\
\text{PEQ}(h_1) & \text{PEQ}(h_2) & \ldots & \text{PEQ}(h_m)
\end{array}$

Notice that $M_{a_i}(f)$ 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 $h_i \Rightarrow M_{a_i}(f)$, because $h_i$ is one of those hypothesis. This implies, $\bigwedge_{i=1}^m h_i \Rightarrow \bigwedge_{i=1}^m M_{a_i}(f) = f$. 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 $|A| \text{Size}_{DNF}(f)$ equivalence queries and $|A| n^2 \text{Size}_{DNF}(f)$ membership queries. We call this algorithm LearnByA.
\[ A = \{ a_1, a_2, \ldots, a_m \} \]

Instead of asking : \( \text{PEQ}(h_1) \quad \text{PEQ}(h_2) \quad \ldots \quad \text{PEQ}(h_m) \)

Ask : \( \text{EQ}(\land h_i) \)

\[
\begin{align*}
    b & \quad \text{if } h_1(b) = 0 \rightarrow b \\
    b & \quad \text{if } h_2(b) = 0 \rightarrow b \\
    b & \quad \text{if } h_m(b) = 0 \rightarrow b
\end{align*}
\]

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 \( \text{PEQ}(h_i) \)

While \( \text{EQ}(\bigwedge_{i=0}^{|A|} h_i) \neq \text{"YES"} \) 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 \( \text{EQ}(h_i) \)

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** \( M_a(f \land g) \Rightarrow M_a(f) \)

*Proof:* According to Lemma 8, \( M_a(f \land g) \Rightarrow M_a(f) \land M_a(g) \Rightarrow M_a(f) \) \( \square \)

**Lemma 12** Let \( C \) be a clause such that \( C(a) = 0 \), then \( M_a(C) = C \)

*Proof:* According to \( M_a \) definition, \( M_a(C)(x) = 1 \) 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, \( M_a(C)(x) = 1 \iff C(x) = 1 \), so \( M_a(C) = C \). \( \square \)
Lemma 13 Let \( A \) be a set of assignments, \( A \subseteq \{0,1\}^n \), and let the CNF representation of some function \( f \) be: \( f_{\text{CNF}} = C_1 \land C_2 \land \ldots \land C_t \). 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 \( a \), \( f \Rightarrow M_a(f) \). Therefore it’s always true that \( f \Rightarrow \wedge_{a \in A} M_a(f) \).

Now, let’s assume that there exists \( a \in A \) such that \( C_i(a) = 0 \), according to Lemma 11 we get \( M_a(f) = M_a(C_1 \land C_2 \land \ldots \land C_t) \Rightarrow M_a(C_i) \), and according to Lemma 12 we get \( M_a(f) \Rightarrow C_i \). Therefore, \( \wedge_{a \in A} M_a(f) \Rightarrow \bigland_{i=1}^t C_i = f \). We saw also that \( f \Rightarrow \bigland_{a \in A} M_a(f) \). So finally we get \( f = \bigland_{a \in A} M_a(f) \) □

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.

\[
EQ(1) \rightarrow a_1 \\
A = \{a_1\} \\
\begin{array}{c}
\text{Run LearnByA} \\
a_2 \text{ negative c.e.} \\
\end{array}
\begin{array}{c}
A = \{a_1, a_2\} \\
a_3 \text{ negative c.e.} \\
\end{array}
\begin{array}{c}
A = \{a_1, a_2, a_3\} \\
\end{array}
\]

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 \( f_{\text{CNF}} \) that hasn’t been falsified yet. So after \( \text{Size}_{\text{CNF}}(f) \) steps, according to Lemma 13, we will have a full basis of \( f \).

Lemma 14 Let \( f = f_1 \land f_2 \) and let \( A = \{a_1, \ldots, a_t\} \) be a set of assignments that falsify all the clauses of \( f_1 \) and not falsify even one clause of \( f_2 \). Let \( h_1, \ldots, h_t \) be the hypothesis we build from \( A \), so \( h_1 \Rightarrow M_{a_1}, \ldots, h_t \Rightarrow M_{a_t} \). And let \( b = a_{t+1} \) be a new negative counterexample, so \( \bigland_{i=1}^t h_i(b) = 1 \) 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 \( h_i \Rightarrow M_{a_i} \), we can say \( \bigland_{i=1}^t h_i \Rightarrow \bigland_{i=1}^t M_{a_i}(f) \).

Now, let \( C_j \) be a clause of \( f_1 \). Notice that \( M_{a_i}(f) = M_{a_i}(C_1 \land C_2 \land \ldots) = M_{a_i}(C_1) \land M_{a_i}(C_2) \land \ldots \Rightarrow M_{a_i}(C_j) \), so \( \bigland_{i=1}^t M_{a_i}(f) \Rightarrow \bigland_{i=1}^t M_{a_i}(C_j) \), and therefore \( \bigland_{i=1}^t h_i \Rightarrow \bigland_{i=1}^t M_{a_i}(C_j) \). 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 \( \bigland_{i=1}^t M_{a_i}(C_j) \Rightarrow M_{a_{i_0}}(C_j) \), and according to lemma 12 we get \( \bigland_{i=1}^t M_{a_i}(C_j) \Rightarrow C_j \), so now we have

\[
\bigland_{i=1}^t h_i \Rightarrow \bigland_{i=1}^t M_{a_i}(f) \Rightarrow \bigland_{i=1}^t M_{a_i}(C_j) \Rightarrow M_{a_{i_0}}(C_j) = C_j
\]

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

14
Finally, the complete algorithm is as follows:

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

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

**Note.** The final output hypothesis is \( \bigwedge_{a \in A} M_a(f) \in \bigwedge \text{DNF} \), 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 $\text{Poly}(n, \text{Size}_{\text{DNF}}(f))$ 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 $\text{Poly}(n, \text{Size}_{\text{DNF}}(f))$ time.

Example:

$O(\log(n)) - \text{termDNF}$

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

$$T_1 \lor T_2 \lor ... \lor T_k = C_1 \land C_2 \land ...$$

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

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

and when $k$ is not a constant the $\text{Size}_{\text{CNF}}(f)$ is not polynomial. But now we will show this class has a small basis. The CNF representation of $f$ is from the $k-\text{CNF}$ 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:

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

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

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

| \ | \ | \ | \ | \ | \ | \ | \ | n \\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>i_1</td>
<td>i_2</td>
<td>...</td>
<td>i_k</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>a_1</td>
<td>a_2</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td></td>
</tr>
<tr>
<td>a_r</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td></td>
</tr>
<tr>
<td>a_m</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td></td>
</tr>
</tbody>
</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 $2^k \log(n) \leq m \leq k2^k \log(n)$ then there exists a $(n,k)$-Universal Set. we will prove only the right inequality which implies that if $k = O(\log(n))$ then $m = \text{Poly}(n)$. 

16
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 \( a_1, \ldots, a_m \). Let’s check the probability that they are not a \((n,k)\)-Universal Set:

\[
\text{Pr}\{\{a_1, \ldots, a_m\} \text{ is not (n,k)-Universal Set}\} \leq \text{Pr}\[\exists 1 \leq i_1 < i_2 < \ldots < i_k \leq n, \exists (d_1, \ldots, d_k) \in \{0, 1\}^k, \forall a : (a_{i_1}, \ldots, a_{i_k}) \neq (d_1, \ldots, d_k)]
\]

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

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

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

\[
= \binom{n}{k} 2^k \left(1 - \frac{1}{2^k}\right)^m \leq \binom{n}{k} 2^k e^{\frac{m}{2^k}} = \delta \quad \square
\]

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