Computational Complexity of Learning Read-Once Formulas over Different Bases

Lisa Hellerstein [1]

Marek Karpinski [2]

TR-91-014

February, 1991

Abstract

We study computational complexity of learning read-once formulas over different boolean bases. In particular we design a polynomial time algorithm for learning read-once formulas over a threshold basis. The algorithm works in time $O(n^3)$ using $O(n^3)$ membership queries. By the result of [Angluin, Hellerstein, Karpinski, 1989] on the corresponding unate class of boolean functions, this gives a polynomial time learning algorithm for arbitrary read-once formulas over a threshold basis with negation using membership and equivalence queries. Furthermore we study the structural notion of nondegeneracy in the threshold formulas generalizing the result of [Heiman, Newman, Wigderson, 1990] on the uniqueness of read-once formulas over different boolean bases and derive a negative result on learnability of nondegenerate read-once formulas over the basis (AND, XOR).

[1]Supported by a grant from the Siemens Corporation and ONR grant N00014-85-K-0445. A portion of this research was done while the author was visiting the Dept. of Computer Science, University of Bonn, and the University of California at Santa Cruz.

[2]Dept. of Computer Science, University of Bonn, 5300 Bonn 1, and the International Computer Science Institute, Berkeley, California. Supported in part by Leibniz Center for Research in Computer Science, by the DFG, Grant KA 673/4-1 and by the SERC Grant GR-E 68297.

1 Introduction

Read-once formulas are boolean formulas over the basis (AND, OR, NOT) in which each variable in the formula appears exactly once. Various papers in learning theory have considered the learnability of read-once formulas in different learning models [A 89], [AHK 89], [HK 89]. Functions expressible by read-once formulas have been shown to have many interesting combinatorial properties [KLNSW 88]. In this paper, we prove some results about classes of read-once formulas over bases other than (AND, OR, NOT).

Angluin, Hellerstein, and Karpinski [AHK 89] showed that read-once formulas over the basis (AND, OR) can be exactly learned in polynomial time using membership queries. In this paper, we extend the result by showing that *read-once formulas over a threshold basis* (which we call *ROTB formulas*) can be learned exactly in polynomial time using membership queries. By the results of Angluin, Hellerstein, and Karpinski [AHK 89], it follows that the corresponding unate class (which includes the class of read-once formulas) can be learned exactly in polynomial time using membership and equivalence queries. [3]

Heiman, Newman, and Wigderson proved in [HNW 90] that each *distinct, non-degenerate*, ROTB formula expresses a unique function. Two formulas are distinct if they are not tree-isomorphic. The definition of Heiman, Newman, and Wigderson is that a ROTB formula is *non-degenerate* if it does not contain either two adjacent AND gates on a root-leaf path, or two adjacent OR gates on root-leaf path.

We generalize this uniqueness result to apply to classes of read-once formulas over boolean bases B satisfying the following four properties:

[3]A similar result has been also obtained independently by T. R. Hancock, cf. [H 90].

1. B is a *symmetric basis* (i.e. each gate of B computes a symmetric function).
2. if f is a function computed by a gate in B, then $\bar{f}$ (the complement of f) is not a monotone function.
3. if f is a function computed by a gate in B, then B does not contain a gate computing $\bar{f}$.
4. f does not contain a gate computing the function $\overline{\text{XOR}}$.

We begin by generalizing the condition of degeneracy to apply to read-once formulas over bases having the above four properties. We say that a read-once formula over such a basis is degenerate if it contains two adjacent gates along a root leaf path that can be "collapsed", i.e. replaced by a single gate from B. We prove that a formula over such a basis is degenerate iff it contains either two ORs, two ANDs, two XORs in adjacent positions along a root-leaf path. (Therefore, our definition of degeneracy includes the definition of Heiman, Newman, and Wigderson [HNW 90].) We end by proving that if B is a basis satisfying the above four properties, then each distinct non-degenerate read-once formula over B expresses a unique function.

2 Preliminaries

The variable set V_n is defined to be $\{X_1, X_2, \ldots, X_n\}$. If V is any subset of V_n, 1_V denotes the vector that assigns 1 to every element of V and 0 to every element of $V_n - V$. Similarly, 0_V is the complement of 1_V — it assigns 0 to every element of V and 1 to every element of $V - V_n$.

Let f be a monotone boolean function of n arguments. A set of variables $V \subset V_n$ is a *minterm* of f if for every vector X that assigns 1 to every variable in S we have

$f(X) = 1$, and this property does not hold for any proper subset S' of S. A set T of variables is a *maxterm* of f if for any assignment y that assigns 0 to all the variables in T we have $f(y) = 0$, and this property does not hold for any proper subset T' of T.

A boolean formula is *unate* if all negations in the formula occur next to the variables, all (other) gates in the formula compute monotone functions, and for every variable x in the formula, either x always occurs with a negation, or it always occurs without a negation.

If f is a read-once formula over any basis B, and X and Y are two variables in f, then we define $lca(X, Y)$ to be the lowest common ancestor of X and Y in f.

3 Learning Read-Once Formulas over the Threshold Basis

Let Th_k^m denote the boolean function on m variables which has the value 1 if at least k of the m variables are set to 1, and which has the value 0 otherwise. The *boolean threshold basis* is the basis containing all gates computing functions of the form Th_k^m. Note that an AND gate with m inputs computes Th_m^m, and an OR gate with m inputs computes Th_1^m.

We present an algorithm for exactly learning ROTB formulas in polynomial time using membership queries.

3.1 Findmin

Every ROTB formula expresses a monotone function. Our algorithm for learning ROTB functions makes repeated use of the standard greedy procedure for finding minterms of a monotone function f defined on V_n, using a membership oracle for f. This procedure takes as input a subset V of V_n known to contain a minterm of f, and outputs a minterm contained in V. The method is a greedy search, removing as many variables from V as possible while preserving the condition that $f(1_V) = 1$. We change the standard procedure slightly by forcing it to test the variables in Q in increasing order of their indices (e.g. X_1 is tested before X_3).

We present the procedure, *Findmin*^{f}, below for the benefit of the reader. The superscript indicates that the procedure uses a membership oracle for f.

Suppose the variables of V are $\{X_{i_1}, \ldots, X_{i_k}\}$, and $i_1 < i_2 < \ldots < i_k$.

```
                    $Findmin^f(V)$

1. $S' := V$.
2. For $j = 1$ to $k$ do:
    (a) Use a membership query to test whether $f(1_{S'-\{X_{i_j}\}}) = 1$.
        If so, then $S' := S' - \{X_{i_j}\}$.
3. Output $S'$.
```

A dual procedure *Findmax*^{f} takes as input a subset Q containing a maxterm of f, and outputs a maxterm contained in Q.

3.2 The basic subroutine

Our algorithm for learning ROTB formulas using membership queries relies on a basic subroutine calles *LcaRootT*^{f}. *LcaRootT*^{f} takes as input a variable X, a minterm S, and a maxterm T, (of the target formula f) such that $S \cap T = \{X\}$. It outputs the set of variables Y in $T - \{X\}$ such that $lca(X, Y)$ is the root of f. A dual subroutine, *LcaRootS*^{f}, finds the set of variables Y in $S - \{X\}$ such that $lca(X, Y)$ is the root of f.

We defer the presentation of theses subroutines to section 3.5.

3.3 Outline of the Algorithm

In this Section we present an outline of the algorithm. We present the full algorithm in Section 4. The algorithm is recursive, and it learns the target formula f depth first.

To begin, the algorithm generates a minterm S and a maxterm T such that $S \cap T = \{X\}$. Suppose the root of f computes Th_k^m and that the inputs to the root are the outputs of the subformulas $f_1, f_2, \ldots, f_m$. Without loss of generality, assume X is a variable of f_1. Because f is read-once, S is composed of minterms of exactly k of $f_1, f_2, \ldots, f_m$ (including f_1). Without loss of generality, assume it is composed of the minterms of $f_1, f_2, \ldots, f_k$. T is composed of the maxterms of $m - k + 1$ of $f_1, f_2, \ldots, f_m$. It is well known that every minterm and maxterm of a formula have a non-empty intersection. Because $S \cap T = \{X\}$, T does not contain maxterms of $f_2, f_3, \ldots, f_k$. It follows that T contains maxterms of $f_{k+1}, f_{k+2}, \ldots, f_m$, and of f_1.

The algorithm calls *LcaRootT*^{f} to find the set T' of variables Y in $T - \{X\}$ such that $lca(X, Y)$ is the root of f. This set is the union of the maxterms of $f_{k+1}, f_{k+2}, \ldots, f_m$ appearing in T. The algorithm then calls *LcaRootS*^{f} to find the set S' which is the union of the minterms of $f_2, f_3, \ldots, f_k$ appearing in S. The projection of f induced by setting the variables in S' to 1, and the variables in T' to 0, is equal to f_1. The algorithm finds f_1 recursively by simulating calls to the membership oracle for f_1 using the oracle for f.

The algorithm then finds the subformulas $f_2, f_3, \ldots, f_k$ as follows. Until all variables in $S - \{X\}$ have appeared in some recursively generated subformula, the algorithm executes the following loop. First it picks some arbitrary Y in $S - \{X\}$, such that Y has not yet appeared in a recursively generated subformula of f. Let $f' \in \{f_2, \ldots, f_m\}$ be the subformula containing Y. The algorithm uses the greedy procedure to generate a maxterm T_Y such that $S \cap T_Y = \{Y\}$. It then uses *LcaRootT*^{f} and *LcaRootS*^{f} on S and T_Y (as it did with S and T) to find a projection of f that is equal to f'. As the final step of the loop, the algorithm finds f' recursively. By counting the number of iterations of this loop, the algorithm learns the value of k.

In a dual way, the algorithm recursively generates $f_{k+1}, \ldots, f_m$ and learns the value of $m - k$.

The algorithm ends by outputing the formula $Th_k^m(f_1, f_2, \ldots, f_m)$.

3.4 Lemmas

The algorithm is based on five lemmas.

Lemma 1 *For any monotone function g, if S is a minterm of g, and X is a variable in S, then there exists a maxterm T of g such that $T \cap S = \{X\}$. Dually, if T is a maxterm, and X is in T, then there exists a minterm S of g such that $T \cap S = \{X\}$.*

Proof: A maxterm of g is a minimal set which has a non-empty intersection with each minterm of g. Consider the set $(V_n - S) \cup X$. This set intersects S because it contains X. Every other minterm S' of S must contain an element not in $S - \{X\}$, because otherwise S' is a subset of S. Hence S' contains a variable in $(V_n - S) \cup \{X\}$. Therefore $(V_n - S) \cup \{X\}$ intersects every minterm, implying that $(V_n - S) \cup \{X\}$ must contain a maxterm. The dual is proved analogously. $\Box$

Lemma 2 *Let f be a ROTB formula defined on the variable set V_n. Let g be a subformula of f, and let Z be the set of variables appearing in g. Let V' be a subset* *of V_n such that $Z \subset V'$, and $f(1_{V'}) = 1$. Let S be the minterm of f output by $Findmin^f(V')$. We prove that if $S \cap Z \neq \emptyset$, then $S \cap Z$ is the minterm of g output by $Findmin^g(Z)$.*

Proof: Consider the execution of $Findmin^f(V')$.

At each iteration of the loop, a variable X_{i_j} is tested (using a membership query) to see whether it should be eliminated from S'.

Assume $S \cap Z = \emptyset$.

In order to show the lemma, it suffices to show the following two facts.

1. $Findmin^f(V')$ tests the variables of Z in the same order as $Findmin^g(Z)$
2. For every X_i in Z, the output of the membership query in $Findmin^f(V')$ that tests X_i is the same as the output of the membership query in $Findmin^g(Z)$ that tests X_i.

Fact 1 follows immediately from the definition of *Findmin*, which specifies that the variables in the input set are tested in increasing order of their indices.

Fact 2 follows from an observation and a claim. The jth iteration of the loop in $Findmin^f(V')$, tests whether X_{i_j} should be included in the output minterm S'. The observation is that if $X_{i_j} \notin Z$, then the value of $S' \cap Z$ at the start of the jth iteration of the loop is the same as the value of $S' \cap Z$ after the iteration. Thus the value of $S' \cap Z$ remains unchanged while *Findmin* tests variables not in Z.

The claim is that if $X_{i_j} \in Z$, then $f(1_{s'-\{X_{i_j}\}})$ (i.e. the value returned by the membership query in the jth iteration of the loop) is equal to $g(1_{S' \cap Z - \{X_{i_j}\}})$. A simple inductive argument combining the observation and the claim proves Fact 2.

We now prove the claim. By assumption $S \cap Z = \emptyset$. Because g is a subformula of f, f is read-once, and S is a minterm of f, S must contain exactly one minterm of g. After every iteration of the loop in $Findmin^f(V')$, S' contains a set which is a superset of S. S contains a minterm of g, and therefore $g(1_{V'}) = 1$. By monotonicity, $g(1_{S'}) = 1$ after every iteration of the loop. If $f(1_{S'-\{X_{i_j}\}}) = 1$, then X_{i_j} is removed from S'. Therefore, $f(1_{S'-\{X_{i_j}\}}) = 1$ implies that $g(1_{S'-\{X_{i_j}\}}) = g(1_{S' \cap Z-\{X_{i_j}\}}) = 1$.

Conversely, suppose $g(1_{S' \cap Z-\{X_{i_j}\}}) = 1$. Then $g(1_{S'-\{X_{i_j}\}}) = 1$. The assignment $1_{S'-\{X_{i_j}\}}$ is obtained from the assignment $1_{S'}$ by changing the setting of the variable X_{i_j} from 1 to 0. Since $g(1_{S'}) = g(1_{S'-\{X_{i_j}\}}) = 1$, changing the assignment of X_{i_j} in $1_{S'}$ from 1 to 0 does not affect the output of g. The formula f is read-once, and g is a subformula of f, so changing the assignment of X_{i_j} in $1_{S'}$ from 1 to 0 does not affect the output of f either. Therefore $f(1_{S'-\{X_{i_j}\}}) = 1$. $\square$

Lemma 3 *Let f be a ROTB formula defined on the variable set V_n. Let S be the minterm of f output by $Findmin^f(V_n)$, and let X be a variable in S. Let T be the maxterm output by $Findmin^f((V_n - S) \cup \{X\})$. If Y is a variable of $T - \{X\}$, and S_Y is the minterm output by $Findmin^f((V_n - T) \cup \{Y\})$, then*

1) *$S_Y - (S_Y \cap S)$ is a minterm of the subformula rooted at the child of lca(X, Y) containing Y.*

2) *$S - (S_Y \cap S)$ is a minterm of the subformula rooted at the child of lca(X, Y) containing X.*

Proof: Consider a gate on the path from X to the root. Suppose the gate computes Th_k^m. T contains maxterms of exactly $m - k + 1$ of the m subformulas whose outputs are inputs to this gate, including the subformula containing X. S contains a minterm of the subformula containing X, and of the remaining $k-1$ subformulas of which T does not contain a maxterm.

Similarly, if we consider a gate on the path from Y to the root computing Th_k^m, T will contain maxterms of exactly $m-k+1$ of the m subformulas, including the subformula containing Y. S_Y will contain a minterm of the subformula containing Y, and of the remaining $k-1$ subformulas of which T does not contain a maxterm.

Let A be a gate which is on the path from $lca(X,Y)$ to the root such that A is not equal to $lca(X,Y)$. S and S_Y contain minterms of the same subformulas (rooted at children of A), and by Lemma 2, they will contain the same minterms of these subformulas. Similarly, S and S_Y will contain the same minterms of the subformulas rooted at children of $lca(X,Y)$ that do not contain X or Y, and for which T does not contain a maxterm. The two parts of the lemma follow easily from these facts. $\square$

Lemma 4 *Let f be a ROTB formula defined on the variable set V_n. Let S be the output of $Findmin^f(V')$ for some $V' \subset V_n$, and let T be the output of $Findmax^f(V'')$ for some $V'' \subset V_n$. Let $S \cap T = \{X\}$. For all Y in $T - \{X\}$, let S_Y be the minterm output by $Findmin^f((V_n - T) \cup \{Y\})$. If there exists a Y in $T - \{X\}$ such that $S \cap S_Y$ is empty, then*

$$\{Y \in T - \{X\} \mid lca(X,Y) = \textit{root of } f\} = \{Y \in T - \{X\} \mid S_Y \cap S = \emptyset\}.$$

If there is no Y in $T - \{X\}$ such that $S \cap S_Y$ is empty, then

$$\{Y \in T - \{X\} \mid lca(X,Y) = \textit{root of } f\} = \{Y \in T - \{X\} \mid \forall Z \in S - (S_Y \cap S),\ S \cup S_Y - \{Z\} \textit{ contains a minterm}\}.$$

Proof: There are two cases.

- Case 1: The root of f is an OR.

  In this case there is at least one variable Y in $T - \{X\}$ such that $lca(X, Y)$ is the root. S_Y is a minterm of the subformula that contains Y and is rooted at a child of the root of f. It follows that $S \cap S_Y = \emptyset$.

  Now let Y be a member of $T - \{X\}$ such that $lca(X, Y)$ is not the root. Let A be the gate which is the child of the root, on the path from X to the root. The gate A is also on the path from Y to the root. Since f is a (non-degenerate) ROTB formula, A is not an OR gate. It follows that S must contain minterms of at least two subformulas rooted at children of A. Only one of these subformulas contains X. Let h be one of the subformulas not containing X. Because $S_Y \cap T = \{Y\}$, S_Y must contain a minterm of h, and by Lemma 2 it will contain the same minterm of h as S. Therefore $S \cap S_Y$ is not empty.

- Case 2: Root of f is not an OR.

  Suppose the root is Th_k^m $(k \neq 1)$. By the same reasoning as in the second part of Case 1, for all Y in $T - \{X\}$, $S \cap S_Y$ is not empty.

  If $lca(X, Y) =$ root, then for all Z in $S \cap S_Y$, $S \cup S_Y - \{Z\}$ contains a minterm, because setting $S \cup S_Y$ to 1 forces $k + 1$ of the subformulas rooted at children of the root of f to 1.

  If $lca(X, Y)$ is not the root, then setting $S \cup S_Y$ to 1 forces exactly k of the subformulas rooted at children of the root to be 1. By Lemmas 2 and 3, $S \cap S_Y$ must contain a minterm of some subformula h rooted at a child of the root of f, such that h does not contain X (or Y). Let Z be a variable in the minterm of h contained in $S \cap S_Y$. Setting $S \cup S_Y - \{Z\}$ to 1 will force only $k-1$ of the wires into the root to 1, because $S \cup S_Y - \{Z\}$ does not contain a minterm of h. Therefore $S \cup S_Y - \{Z\}$ does not contain a minterm of f. $\Box$

The duals of the above lemmas also hold.

We present the basic subroutines *LcaRootS* and *LcaRootT*, and then we present the complete algorithm.

3.5 LcaRootT and LcaRootS

LcaRootT takes as input a minterm S, a maxterm T, and a variable X, such that $S \cap T = \{X\}$. S is the output of $Findmin^f(V'')$, where V'' is a subset of V_n.

The output of *LcaRootT* is the set of variables Y in $T - \{X\}$ such that $lca(X, Y)$ is the root of f.

$$LcaRootT^f(S, T, X)$$

```
1. for all $Y$ in $T - \{X\}$        $S_Y := Findmin^f((V_n - T) \cup \{Y\})$.

2. if there exists a $Y$ in $T - \{X\}$ such that $S \cap S_Y$ is empty then return( $\{Y | S_Y \cap S = \emptyset\}$ ).

3. $Q := \emptyset$

        for all $Y$ in $T - \{X\}$ do
           for all $Z$ in $S \cap S_Y$ do
              if $f(1_{S \cup S_Y - \{Z\}}) = 0$ then
                 $Q := Q \cup \{Y\}$.
```

4. Output $T - \{X\} - Q$

A dual subroutine finds the set of Y in $S - \{X\}$ such that $lca(X, Y)$ is the root of f.

4 The Algorithm

$ROTBLearn^f$

```
1. $S := Findmin^f(V_n)$
2. Pick an $X$ in $S$.
   $T := Findmax^f((V_n - S) \cup \{X\})$
3. if $S = T = \{X\}$, then return($X$) (the formula $f$ is equal to $X$)
4. $T' := LcaRootT^f(S, T, X)$
5. $S' := LcaRootS^f(S, T, X)$
6. (a) $k := 1$ (counts number of inputs to root of $f$ set to 1 by a minterm of $f$)
   (b) $j := 1$ (counts number of inputs to root of $f$ set to 0 by a maxterm of $f$)
   (c) $Q := S'$
   (d) $R := T'$
   (e) Let $f_1$ be the projection of $f$ induced by setting the variables in $S'$
       to 1, and the variables in $T'$ to 0. Recursively learn $f_1$ by running
       $ROTBLearn^{f_1}$, simulating calls to the membership oracle of $f_1$ with calls
       to the membership oracle of $f$.
```

```
7. while $Q \neq \emptyset$ do
    (a) Pick an $X'$ in $Q$.
    (b) $T_{X'} := \mathit{Findmax}^f((V_n - S) \cup \{X'\})$
    (c) $S' := \mathit{LcaRootS}^f(S, T_{X'}, X')$
    (d) $k := k + 1$.
    (e) $Q := Q \cap S'$.
    (f) Let $f_k$ be the projection of $f$ induced by setting the variables in $S'$
        to 1, and the variables in $T'$ to 0. Recursively learn $f_k$ by running
        $\mathit{ROTBLearn}^{f_k}$, simulating calls to the membership oracle of $f_k$ with
        calls to the membership oracle of $f$.
8. while $R$ not empty do
    (a) Pick an $X'$ in $R$.
    (b) $S_{X'} := \mathit{Findmin}^f((V_n - T) \cup \{X'\})$
    (c) $T' := \mathit{LcaRootT}^f(S'_X, T, X')$.
    (d) $j := j + 1$.
    (e) $R := R \cap T'$.
    (f) Let $f_{k+j-1}$ be the projection of $f$ induced by setting the variables in
        $S'_X \cap S$ to 1, and the variables in $T'$ to 0. Recursively learn $f_{k+j-1}$ by
        running $\mathit{ROTBLearn}^{f_{k+j-1}}$, simulating calls to the membership oracle of
        $f_{k+j-1}$ with calls to the membership oracle of $f$.
9. Output the formula $Th_k^{k+j-1}(f_1, f_2, f_3, \cdots, f_{k+j-1})$
```

5 Correctness and Complexity

Theorem 1 *There is a learning algorithm that exactly identifies any ROTB formula in time $O(n^3)$ using $O(n^3)$ membership queries.*

Proof: Consider the algorithm described in the above sections. The correctness of the algorithm follows from the five lemmas proved in Section 3.4.

The routines *Findmin* and *Findmax* each take time $O(n)$ and make $O(n)$ queries. The routine *LcaRootT* makes $O(n^2)$ queries and can be implemented to run in time $O(n^2)$ (this includes the calls to *Findmin*).

The complexity of the main algorithm can be calculated by "charging" the costs of the steps to the edges and nodes of the target formula f. In each execution of $ROTB^f$, we charge some of the steps to f, and some of the steps to the edges joining the root to its children. Recursive calls to $ROTB^{f_k}$ are charged recursively to the subformula f_k.

More specifically, we charge steps 1 - 6(d), step 9, and the checking of the loop conditions in steps 7 and 8, to the root of f. We recursively charge calls to $ROTBLearn^{f_k}$ in steps 6(e), 7(f), and 8(f) to the subformulas f_k. For each iteration of step 7, we charge steps 7(a) through 7(e) to the edge leading from the root of f to the root of the subformula f_k defined in step 7(f). Similarly, for each iteration of step 8, we charge steps 8(a) through 8(e) to the edge leading from the root of f to the root of the subformula f_{k+j-1} defined in step 8(f). Thus at each execution of $ROTB^f$, we charge time $O(n^2)$ to the root of f and $O(n^2)$ membership queries to the root of f. We also charge time $O(n^2)$ and $O(n^2)$ membership queries to each of the edges joining the root of f to its children.

The total number of nodes in f is $O(n)$, and the total number of edges is $O(n)$. Therefore the algorithm takes time $O(n^3)$ and makes $O(n^3)$ queries. $\Box$

Corollary 1.1 *There is a learning algorithm that exactly identifies any read-once formula over the basis consisting of the threshold formulas and NOT (ROTB with negations) in time $O(n^4)$ using $O(n^4)$ membership queries and $O(n)$ equivalence queries.*

Proof: The class of formulas mentioned in this theorem is the unate extension of the class of ROTB formulas. The theorem follows directly from the results of Angluin, Hellerstein and Karpinski [AHK 89], who showed that if a class M can be learned in time $O(n^k)$ with $O(n^j)$ membership queries, then the corresponding unate class can be learned in time $O(n^{k+1})$ with $O(n^{j+1})$ membership queries, and $O(n)$ equivalence queries. $\Box$

6 The uniqueness of read-once formulas

6.1 Definitions

A boolean formula over a symmetric basis B is *degenerate* if it contains two gates C and D appearing on adjacent levels, D's output is an input to C, such that C computes the function $f(x_1, x_2, \cdots, x_k)$, D computes the function $g(y_1, y_2, \cdots, y_m)$, and the function $h(z_1, z_2, \cdots, z_{m+k-1}) = f(g(z_1, z_2, \cdots, z_m), z_{m+1}, z_{m+2}, \cdots, z_{m+k-1})$ is computed by a gate in B.

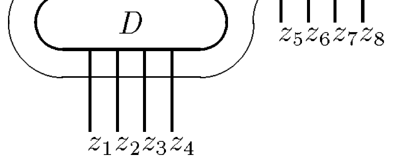

Figure 0

In a degenerate formula, the gates C and D could be replaced by a gate Q computing the function h.

Definition: A boolean function f is *monotonically-decreasing* if the function $\bar{f}$ is monotone.

6.2 Conditions for degeneracy

Theorem 2 *A boolean formula over a symmetric basis of non-constant, non-monotonically-decreasing functions, not containing the function* $\overline{\text{XOR}}$*, is degenerate if and only if it contains at least one of the following pairs of gates on adjacent levels:*

1. *OR, OR*
2. *XOR, XOR*
3. *AND, AND*

Proof: It is clear that if a formula contains any one of the pairs of gates in the above list on adjacent levels, then it is degenerate.

We now prove that the absence of these pairs of gates implies that the formula is not degenerate.

Let $f(x_1, \cdots, x_k)$ and $g(y_1, \cdots, y_m)$ be two symmetric functions in a basis B satisfying the conditions of the theorem, and not appearing as a pair in the above list. Let $h(z_1, z_2, \cdots, z_m, z_{m+1}, \cdots, z_{m+k-1}) = f(g(z_1, z_2, \cdots, z_m), z_{m+1}, z_{m+2}, \cdots, z_{m+k-1})$. We prove that h is not a symmetric function, and therefore h is not computed by a gate in B.

Because f and g are symmetric functions, their outputs depend only upon the number of input variables that are set to 1. By an abuse of notation, let $f(q)(0 \leq q \leq k)$ be the value of f on an input assignment with exactly q input variables set to 1. Define $g(q)$ similarly.

- Case 1: f is not AND or OR, and g is not XOR.

  The functions f and g are symmetric and so their values depend only on the number of input variables that are set to 1. g is not XOR (and by assumption it's not $\overline{\text{XOR}}$), so there exists an r such that $0 \leq r \leq m-1$, and $g(r) = g(r+1)$.

  Suppose $g(r) = g(r+1) = 1$. f is not monotonically-decreasing and not OR, so there is some q' such that $1 \leq q' \leq k-1$, $f(q') = 0$, and $f(q'+1) = 1$. Then $f(g(1^r, 0^{m-r}, 1^{q'}), 0^{k-(q'+1)}) = f(q'+1)$ and $f(g(1^{r+1}, 0^{m-r}), 1^{q'-1}, 0^{k-q'}) = f(q') = 0$ (where 0^i denotes i 0's separated by commas, and 1^r denotes r 1's separated by commas), so h is not symmetric.

  Conversely, suppose $g(r) = g(r+1) = 0$. We first claim that there exists a q such that $0 \leq q \leq k-2$ and $f(q) \neq f(q+1)$.

  Suppose not. Then $f(0) = f(1) = \cdots = f(k-1)$. f is not a constant function, so $f(k) \neq f(k-1)$. It follows that either $f(0) = \cdots = f(k-1) = 0$ and $f(k) = 1$, or $f(0) = \cdots = f(k-1) = 1$ and $f(k) = 0$. In the first case f is

AND, and in the second case f is NAND, which is a monotonically-decreasing function. Contradiction. Therefore, there exists a q such that $0 \leq q \leq k-2$ and $f(q) \neq f(q+1)$. Then $f(g(1^r, 0^{m-r}), 1^{q+1}, 0^{k-(q+2)}) = f(q+1)$. But $f(g(1^{r+1}, 0^{m-(r+1)}), 1^q, 0^{k-(q+1)}) = f(q)$, and so h is not symmetric.

- Case 2: f is AND.

  By assumption, in this case g cannot be AND.

  If there exists an $r \in [0..m-1]$ such that $g(r) = 1$ and $g(r+1) = 0$, then $f(g(1^r, 0^{m-r}), 1^{k-1}) = f(k) = 1$, but $f(g(1^{r+1}, 0^{m-(r+1)}), 1^{k-2}, 0) = f(k-2) = 0$, so h is not symmetric.

  If there is no $r \in [0..m-1]$ such that $g(r) = 1$ and $g(r+1) = 0$, then g is monotone. Because g is non-constant but not AND, $g(m-1) = g(m) = 1$. Therefore $f(g(1^m), 1^{k-2}, 0) = f(k-1) = 0$, but $f(g(1^{m-1}, 0), 1^{k-1}) = f(k) = 1$, and h is not symmetric.

- Case 3: f is OR.

  By assumption, in this case g cannot be OR.

  If there exists an $r \in [0..m-1]$ such that $g(r) = 1$ and $g(r+1) = 0$, then $f(g(1^r, 0^{m-r}), 1, 0^{k-2}) = f(2) = 1$, but $f(g(1^{r+1}, 0^{m-(r+1)}), 0^{k-1}) = f(0) = 0$, so h is not symmetric.

  If there is no $r \in [0..m-1]$ such that $g(r) = 1$ and $g(r+1) = 0$, then g is monotone. Because g is non-constant but not OR, $g(0) = g(1) = 0$. Therefore $f(g(1, 0^{m-1}), 0^{k-1}) = f(0) = 0$, but $f(g(0^m), 1, 0^{k-2}) = f(1) = 1$, and h is not symmetric.

- Case 4: g is XOR.

  In this case f cannot be XOR (and by assumption, it's not $\overline{\text{XOR}}$). Because f cannot be constant either, there must exist a $q \in [0..k-2]$ such that $f(q) \neq f(q+2)$. g is XOR, so $f(g(1, 0^{m-1}), 1^{q+1}, 0^{k-(q+2)}) = f(q+2)$ and $f(g(1, 1, 0^{m-2}), 1^q, 0^{k-(q+1)}) = f(q)$, so h is not symmetric. $\Box$

Theorem 3 *Let B be a symmetric basis of non-constant, non-monotonically-decreasing functions, not containing the function $\overline{XOR}$. Then every non-degenerate read-once formula over B expresses a distinct function.*

Proof: Let f be a non-degenerate read-once formula over a symmetric basis B of non-constant, non-monotonically-decreasing functions not containing the function $\overline{\text{XOR}}$. Let f be defined on the variable set $X = \{x_1, x_2, \cdots, x_n\}$. The proof is by induction on the number of gates and input wires in f. When f consists of just one input wire, the theorem clearly holds. We now prove the inductive step.

Because each gate in f is non-monotonically-decreasing, an easy induction argument on the number of gates in f proves that for any $x_i \in \{x_1, \cdots, x_k\}$, there exist two assignments $A : X \longrightarrow \{0,1\}^k$ and $A' : X \longrightarrow \{0,1\}^k$ such that $f(A) = 0$, $f(A') = 1$, $A(x_i) = 0$, $A'(x_i) = 1$, and for all $j \neq i$, $A(x_j) = A'(x_j)$.

We say that two variables x_i and x_j in X are *symmetric with respect to f* if changing the positions of the variables x_i and x_j in the formula f does not affect the function computed by the formula. That is, x_i and x_j are symmetric with respect to f if for every assignment A to X, the values of $f(x_1, x_2, \cdots, x_{i-1}, x_i, x_{i+1}, \cdots, x_{j-1}, x_j, x_{j+1}, \cdots, x_k)$ and $f(x_1, x_2, \cdots, x_{i-1}, x_j, x_{i+1}, \cdots, x_{j-1}, x_i, x_{j+1}, \cdots, x_k)$ are equal on assignment A.

We now prove Claim 1.

Claim 1: x_i and x_j are symmetric with respect to f iff x_i and x_j are children of a common node in f. If x_i and x_j are children of a common node in f, then they are clearly symmetric with respect to f. Conversely, suppose that x_i and x_j are not children of a common node. Then the lowest common ancestor is a gate computing a function $g(w_1, w_2, \cdots, w_s)$. One of the inputs to the gate (say w_1) is a formula p that depends on x_i, and another (say w_2) is a formula q that depends on x_j. (Figure 1)

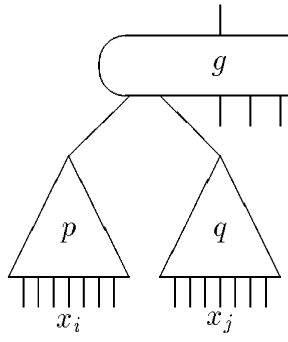

Figure 1

Because x_i and x_j are not children of a common node, at least one of p and q contains at least one gate. Without loss of generality, assume p contains at least one gate. Let $g_1(y_1, \cdots, y_r)$ be the function computed by the gate at the root of p. Let p' be the subformula of p that is rooted at a child of the root of p, and contains the variable x_i. (Figure 2)

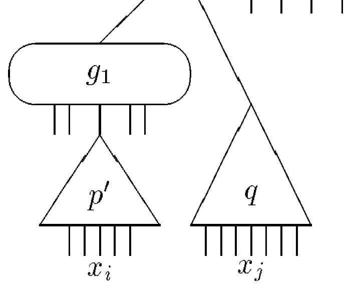

Figure 2

Let $h(z_1, \cdots, z_{r+s-1}) = g(g_1(z_1, \cdots, z_r), z_{r+1}, \cdots, z_{r+s-1})$. The formula f is not degenerate, so $h(z_1, \cdots, z_{r+s-1})$ is not symmetric. Therefore, there exists an assignment $A : \{z_1, \cdots, z_{r+s-1}\} \longrightarrow \{0, 1\}$ such that $A(z_1) = 0$, $A(z_{r+1}) = 1$, and the value of $g(g_1(z_1, z_2, \cdots, z_r), z_{r+1}, z_{r+2}, \cdots, z_{r+s-1})$ under the assignment A is not equal to the value of $g(g_1(z_{r+1}, z_2, \cdots, z_r), z_1, z_{r+2}, \cdots, z_{r+s-1})$ under the assignment A. We have already argued that there must exist two assignments B and B' to the variables in p' such that $B(x_i) = 0$ and $B'(x_i) = 1$, B and B' differ only in their assignment to x_i, and $p'(B) = 0$ while $p'(B') = 1$. Similarly, there are two assignments C and C' to the variables in q such that $C(x_i) = 0$ and $C'(x_i) = 1$, C and C' differ only in their assignment to x_i, and $f(C) = 0$ while $f(C') = 1$.

The assignment B sets x_i to 0, and the output of p' (which is the input z_1 to the formula h) to 0. The assignment C' sets x_j to 1, and the output of q (which is the input z_{r+1} to the formula h) to 1. The assignment B and C' can be extended to the other variables of f in such a way that all inputs $z_1 \cdots z_{r+s-1}$ to h are set according to the assignment A. Call this new assignment D. Define a new assignment D' which is the same as D except that the value of x_i is 1, and the value of x_j is 0. The assignment D' sets the inputs $z_1 \cdots z_{r+s-1}$ to h according to the assignment A. Therefore, x_i and x_j are not symmetric with respect to the subformula of f rooted at g. Because f depends on the output of g, it follows that x_i and x_j are not symmetric with respect to f. This completes the proof of Claim 1.

It follows from the claim that given the truth table of f it is possible to determine which variables of f are children of the same node (i.e. siblings).

Consider a set of variables Y that are siblings, and whose parent has no other children besides these variables. Let g be the parent gate. The value of f depends on the output of g. Consider any assignment to the variables in $X - Y$. The induced function on Y is either $g, \bar{g}$, or a constant. Furthermore, there exists some assignment to the variables in $X - Y$ such that the induced function is not constant. (Figure 3)

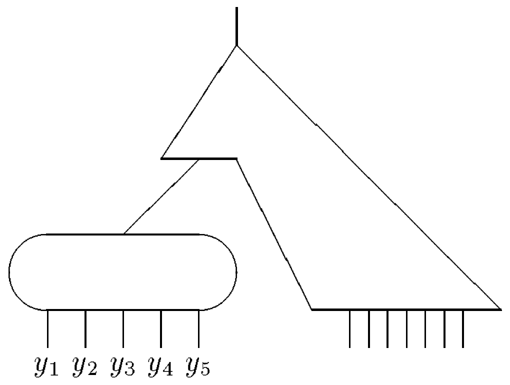

Figure 3

Now consider a set of (at least 2) variables Y that are siblings, and whose parent gate g has children other than Y. Consider the induced functions formed by assigning values to the variables in $X - Y$. It follows directly from Claim 2 (proved below) that if g is not XOR, AND, $\overline{\text{XOR}}$ or OR, then the induced functions include two distinct functions f_1 and f_2 such that f_1 and f_2 are not constant, and $f_1 \neq \overline{f_2}$. Thus if the parent of the variables in Y is not one of the above gates, then the truth table of f will reveal that g has children other than the variables in Y. (Figure 4)

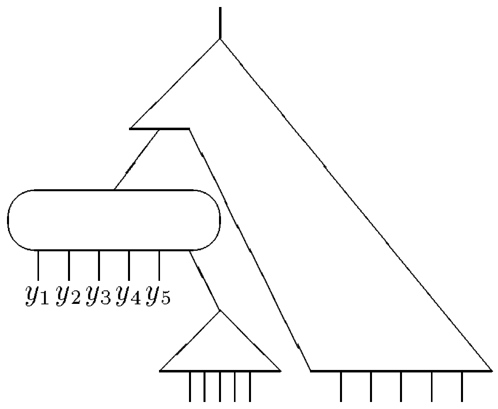

Figure 4

The uniqueness of the read-once formula f can be proved by the following argument. f must have some gate g such that all of g's inputs are input wires (variables). By examining the truth table of f, we determine which variables are siblings. For every set Y that is a maximal set of siblings, we examine the induced functions formed by assigning values to $X-Y$. If the induced functions include two distinct nonconstant functions f_1 and f_2 such that $f_1 \neq \overline{f}_2$, then we know that g has children that are not input wires (variables). Otherwise we know that either g is AND, OR, or XOR, or g has no children other than the variables in Y.

Ther must exist some set of siblings Y whose parent g has no children other than the variables in Y. Given the truth table of f, we can find one such set of variables Y, and examine the induced functions formed by assigning values to X and Y. The induced functions include only g, $\overline{g}$, and constants.

The basis from which f is formed does not contain one gate computing a function, and another computing its complement. Thus by examining the induced functions on Y we can uniquely determine g. If g is not AND, OR, or XOR, (by assumption it is not $\overline{\text{XOR}}$) then consider the read-once formula f' obtained from f by replacing the gate g by a new input variable (wire) v. The truth table of f' is determined by f and g. By induction, f' is unique. f can be reconstructed from f' by replacing v with g and its children. Therefore f is unique. If g is AND, OR, or XOR, then consider the read-once formula f' obtained from f by replacing the input wires for the variables in Y with a single input wire for a new variable v (and eliminating g if this causes g to have just one input). The truth table of f' is also determined by f and g in this case. Given f', we can reconstruct f as follows. If the parent of v is not g, we replace v with g and its children (the variables in Y were the only children of g). Otherwise, we replace v with the input wires for the variables in Y (the parent of v is the parent of the variables in Y).

We now prove Claim 2.

Claim 2: Let g be a non-constant, non-monotonically-decreasing, symmetric, boolean function not computing OR, AND, XOR, or $\overline{\text{XOR}}$. Let g be defined on the variable set $X = \{x_1, \cdots, x_n\}$, where $n \geq 3$. Let $Y \subset X$ such that $|Y| \geq 2$. Then there exist two assignment P and P' to the variables of $X - Y$ such that the functions g_P and $g_{P'}$ induced on g by P and P' are not constant, and $g_P \neq \overline{g_{P'}}$.

Proof: We first argue that it suffices to prove the above theorem for the case $|Y| = 2$. Suppose the theorem holds for $|Y| = 2$. If $|Y| > 2$, then let Y' be a subset of Y such that $|Y'| = 2$. Thus there exist assignments P and P' to the variables in $X - Y'$ such that g_P and $g_{P'}$ are not constant, and $g_P \neq \overline{g_{P'}}$. Let Q be the assignment to the variables in $X - Y$ that sets the variables in $X - Y$ precisely according to P (but leaves the variables in $Y - Y'$ unassigned). g_P and $g_{P'}$ are projections of g_Q and $g_{Q'}$ induced by assigning values to the variables in $Y - Y'$. Since g_P and $\overline{g_{P'}}$ are not constant, and $g_P \neq \overline{g_{P'}}$, it follows that g_Q and $g_{Q'}$ are not constant, and $g_Q \neq \overline{g_{Q'}}$.

By the above argument, we can assume $|Y| = 2$. g is a symmetric function. With every symmetric function on n variables, we can associate a binary string $a_0 a_1 a_2 \cdots a_n$ in which bit a_i is the output of the function when exactly i of the variables of the n variables are set to 1, and the others are set to 0. If g_1 and g_2 are two symmetric functions on n variables, then $g_1 = \overline{g}_2$ if and only if the two strings are complementary (i.e. each bit in the binary string associated with g_1 is the complement of the corresponding bit in the string associated with g_2).

Let $K = k_0 k_1 \cdots k_n$ be the string associated with the function g. Consider the assignment P setting exactly i of the variables in $X - Y$ to 1, and setting the other variables in $X - Y$ to 0. Let g_P be the function induced by P. The binary string associated with g_P is $k_i k_{i+1} k_{i+2}$. It follows that Claim 2 holds iff there exist 2 distinct substrings $k_i k_{i+1} k_{i+2}$ and $k_j k_{j+1} k_{j+2}$ of K such that neither of the substrings is all 0's or all 1's, and the two strings are not complementary. We now prove this is indeed true. The proof is by case analysis.

Case 1: $k_0 = 1$ and $k_1 = 1$

g is not constant or monotonically-decreasing, so K is a member of the language expressed by the regular expression $11^{+}00^{*}1\{0,1\}^{*}$. Therefore K contains the substring 110, and one of the two substrings 100 and 101.

110 is not the complement of either 100 or 101.

Case 2: $k_0 = 1$ and $k_1 = 0$

g is not $\overline{\text{XOR}}$, and it is non-monotonically-decreasing. K is therefore a member of one of the following languages:

- 1000*1 {0,1}: K contains 100 and 001
- 1011{0,1}*: K contains 101 and 011
- 1010{10}*0{0,1}*: K contains 101 and 100
- 1010{10}*11{0,1}*: K contains 101 and 011

Case 3: $k_0 = 0$ and $k_1 = 0$

g is not constant or AND, so K is in one of the following two languages:

- 000*11{0,1}*: K contains 001 and 011
- 000*10{0,1}*: K contains 001 and 010

Case 4: $k_0 = 0$ and $k_1 = 1$

g is not XOR or OR, so K is in one of the following four languages:

- 0111*0{0,1}*: K contains 011 and 110
- 0100{0,1}*: K contains 010 and 100
- 0101{01}*1{0,1}*: K contains 010 and 011
- 0101{0,1}*00{0,1}*: K contains 010 and 100

This complete the proof of Claim 2, and of the theorem. □

We have restricted our discussion to bases not including monotonically-decreasing functions. Non-degenerate read-once formulas over bases containing monotonically-decreasing functions are not necessarily unique. For instance, OR(NAND(x, y), NAND(w, z)) is non-degenerate and equal to NAND(x, y, w, z).

Non-degenerate read-once formulas over the basis (AND, OR, NOT) are not unique unless you add the restriction that the negations must be at the leaves.

The function $\overline{\mathrm{XOR}}$ presents a related problem. Note that over the basis (XOR, AND), for example, the formula $\overline{\mathrm{XOR}}(a, (\overline{\mathrm{XOR}}(b(\overline{\mathrm{XOR}}(c, d)))))$ is not degenerate, and it is equal to XOR(a, b, c, d). An additional problem with the function $\overline{\mathrm{XOR}}$ is that two $\overline{\mathrm{XOR}}$'s on adjacent levels of a formula compute the XOR of all their inputs. Thus if $\overline{\mathrm{XOR}}$ is included in a basis, its complement XOR is also, in effect, included in the basis. It is very easy to come up with examples in which having both a function and its complement in a basis leads to distinct non-degenerate read-once formulas that compute the same function.

In general, then, in order to prove uniqueness for bases which don't have the properties listed in the theorem, it would be necessary to introduce other conditions in addition to non-degeneracy. It is trivial to prove uniqueness if enough conditions are introduced (e.g. that the formula is the lexicographically smallest of all read-once formulas over the basis that compute the same function). We believe that the kinds of conditions that are interesting are those that have the following property: Any read-once formula over the given basis that does not satisfy the conditions can be transformed in polynomial time into a read-once formula over the same basis that satisfies the conditions.

7 Uniqueness and learnability

The proof of Theorem 3 in the previous section gives an algoritm for learning read-once formulas over non-monotonically-decreasing, non-constant, symmetric bases not containing $\overline{\mathrm{XOR}}$, given the truth table of the formula. Unfortunately, the algorithm takes time exponential in n. The algorithm we give for learning ROTB formulas differs substantially from this truth table algorithm. One difference is that it learns the formula top-down, rather than bottom-up.

Our algorithm for learning ROTB formulas in polynomial time with membership queries exploits the structure of the target formula. The fact that the formula is unique allows us to do this. However, as we prove in the next theorem, not all classes of read-once formulas over bases of the type treated in Theorem 3 can be learned in polynomial time using membership queries alone.

Theorem 4 *The class of non-degenerate read-once formulas over the basis (AND, XOR) cannot be learned in polynomial time using membership queries alone.*

Proof: Let $X = \{x_1, x_2, \cdots, x_n\}$ be a set of variables such that n is even.

Consider the class of read-once formulas over the basis (AND, XOR) that have the form $\mathrm{AND}((x_{i_1}\mathrm{XOR}x_{i_2}), (x_{i_3}\mathrm{XOR}x_{i_4}), (x_{i_5}\mathrm{XOR}x_{i_6}), \cdots, (x_{i_{n-1}}\mathrm{XOR}x_{i_n}))$. There are $n!/((n/2)!\,2^{n/2})$ distinct formulas in this class. The satisfying assignments of a formula in this class set exactly $n/2$ of the variables of X to 1. Consider an arbitrary assignment that sets exactly $n/2$ of the variables of X to 1. This assignment will be a satisfying assignment for exactly $(n/2)!$ of the formulas in the above class.

Assume there is a polynomial time algorithm for learning the set of read-once formulas over the basis (XOR, AND) using membership queries.

Consider the following adversary strategy. Each time the algorithm queries the membership oracle on an input that does not have exactly $n/2$ 1's, give the answer 0. Each time the algorithm queries the membership oracle on an input that has exactly $n/2$ 1's, give the answer 0, also.

Since the algorithm runs in polynomial time, it makes at most n^k queries for some constant k. After n^k queries, the total number of formulas in the above class that do not agree with the answers given by the adversary is at most $n^k(n/2)!$. The total number of formulas in the above class that agree with the answers is at least $n!/((n/2)!\,2^{n/2}) - n^k(n/2)! > 1$ for large enough n. Therefore, the algorithm cannot distinguish between all formulas in the above class using n^k queries. Contradiction. $\square$

8 Open Problems

It is an open question whether for every basis B of the type treated in Theorem 3, there exists a polynomial time algorithm for learning the read-once formulas over B using both membership and equivalence queries.

Another problem is to improve the complexity bounds of Theorem 1, and Corollary 1.1 on learning read-once threshold formulas.

References

[A 87a] D. Angluin, *Learning k-term DNF Formulas Using Queries and Counterexamples*, Technical Report, Yale University, YALE/DCS/RR-559, 1987.

[A 87b] D. Angluin, *Learning Regular Sets from Queries and Counterexamples*, Information and Computation, 75:87-106, 1987.

[A 88] D. Angluin, *Queries and Concept Learning*, Machine Learning, 2:319-342, 1988.

[A 89] D. Angluin, *Using Queries to Identify μ-Formulas*, Technical Report, Yale University, YALE/DCS/RR-694, 1989.

[AHK 89] D. Angluin, L. Hellerstein, and M. Karpinski, *Learning Read-Once Formulas with Queries*, U. C. Berkeley Technical Report UCB CSD 89-258 and International Computer Science Institute Technical Report TR 89-021, 1989, to appear in J. ACM (1991).

[BI 87] M. Blum and R. Impagliazzo, *Generic Oracles and Oracle Classes*, in Proc. 28^{th} IEEE Symposium on Foundations of Computer Science, pages 118-126. IEEE, 1987.

[H 90] T. R. Hancock, *Identifying μ-Formula Decision Trees with Queries*, Technical Report TR-16-90, Aiken Computation Laboratory, Harvard University, 1990.

[HK 89] L. Hellerstein and M. Karpinski, *Learning Read-Once Formulas Using Membership Queries*, in Proc. of the Second Annual Workshop on Computational Learning Theory, pages 146-161. Morgan Kaufmann Publishers, 1989.

[HNW 90] R. Heiman, I. Newman, A. Wigderson, *On Read Once Threshold Formulas and their Randomized Decision Tree Complexity*, IEEE Symp. on Structures in Complexity 1990, pp. 78-87.

[KLNSW 88] M. Karchmer, N. Linial, I. Newman, M. Saks, and A. Wigderson, *Combinatorial Characterization of Read Once Formulae*, Presented at the Joint French-Israeli Binational Symposium on Combinatorics and Algorithms, 1988. To appear in Discrete Math.

[KLPV 87] M. Kearns, M. Li, L. Pitt, and L. Valiant, *On the Learnability of Boolean Formulae*, in Proc. 19^{th} ACM Symposium on Theory of Computing, pages 285-295. ACM, 1987.

[KV 89] M. Kearns and L. Valiant, *Cryptographic Limitations on Learning Boolean Formulae and Finite Automata*, in Proc 21^{st} ACM Symposium on Theory of Computing, pages 433-444. ACM, 1989.

[LMN 89] N. Linial, Y. Mansour, and N. Nisan, *Constant Depth Circuits, Fourier Transform, and Learnability*, in Proc 30^{th} IEEE Symposium on Foundations of Computer Science, pages 574-579. IEEE, 1989.

[PV 89] L. Pitt and L. Valiant, *Computational Limitations on Learning from Examples*, J. ACM, 35:965-984, 1988.

[V 84] L. G. Valiant, *A Theory of the Learnable*, C. ACM, 27:1134-1142, 1984.