Read-Once Threshold Formulas, Justifying Assignments, and Generic Tranformations

Nader H. Bshouty[1]
Thomas R. Hancock[2]
Lisa Hellerstein[3]
Marek Karpinski[4]

TR-92-020

March, 1992

Abstract

We present a membership query (i.e. interpolation) algorithm for exactly identifying the class of read-once formulas over the basis of boolean threshold functions. Using a generic transformation from [Angluin, Hellerstein, Karpinski 89], this gives an algorithm using membership and equivalence queries for exactly identifying the class of read-once formulas over the basis of boolean threshold functions and negation. We also present a a series of generic transformations that can be used to convert an algorithm in one learning model into an algorithm in a different model.

---

[1]Department of Computer Science, The University of Calgary, Calgary, Alberta, Canada T2N 1N4. This research was supported in Part by the NSERC of Canada.

[2]Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138. Supported by ONR grant N00014-85-K-0445.

[3]Department of EECS, Northwestern University, Evanston, IL 60208-3118. Supported in part by a grant from the Siemens Corporation and ONR grant N00014-85-K-0445. Portions of this research were done while the author was visiting the Dept. of Computer Science, University of Bonn, the University of California at Santa Cruz, and the Massachussetts Institute of Technology.

[4]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

The technical content of this paper can be divided into two parts. In the first part, we present query algorithms for learning read-once formulas over the basis of boolean threshold functions and negation. In the second, we present a series of generic transformations that can be used to convert an algorithm in one learning model (the *base model*) into an algorithm in a different model (the *target* model). Such transformations can be regarded as reductions, in that they reduce the problem of designing an algorithm in one learning model, into the problem of designing an algorithm in another learning model.

The results in this paper generalize the work of Angluin, Hellerstein, and Karpinski on learning read-once formulas [AHK 89]. A read-once formula is a boolean formula over the basis (AND,OR,NOT) that contains at most one occurrence of each variable. Angluin, Hellerstein, and Karpinski gave a polynomial time membership query algorithm for exactly identifying (i.e. interpolating) the class of monotone read-once formulas, where the basis is just (AND,OR). Then they applied a generic transformation to produce a polynomial time membership and equivalence query algorithm for the non-monotone case. It is easily shown that there is no membership query only algorithm for the non-monotone case.

We generalize the monotone [AHK 89] algorithm by presenting a polynomial time membership query algorithm to exactly identify read-once formulas over the basis of boolean threshold functions (a superset of the basis (AND,OR)). The [AHK 89] transformation then implies a membership and equivalence query algorithm to identify read-once formulas over the basis of boolean threshold and negation.

The class of read-once formulas over the basis of boolean threshold and negation was previous studied by Heiman, Newman, and Wigderson [HNW 90] . They proved that each *non-degenerate* read-once formula over this basis expresses a unique function (this result follows implicitly from our learning algorithm), and proved bounds on the size of randomized decision trees computing such functions.

The second part of this paper demonstrates a more extensive set of generic transformations. As in the [AHK 89] transformation, in these transformations we assume the technical condition that the classes being learned are projection closed. In our transformations, either the base or target model has the property that the learning algorithm (in addition to being able to ask queries) is given as input a set of "justifying assignments" for the relevant variables. A justifying assignment for a variable x in a formula f is an assignment in which flipping the value of x would change the output of f. We say that algorithms in such models "use justifying assignments" (meaning that they receive them as input) just as we say that they use certain types of queries. Many of the transformations convert algorithms in models that use justifying assignments into algorithms in models that do not, and thus can be useful in some cases for simplifying the process of algorithm design.

Our interest in these transformations began with an alternative membership and equivalence query algorithm for identifying read-once formulas over the basis of threshold functions and negations [Han 90]. This algorithm is qualitatively different from the one described in the first part of this paper, but achieves the same bounds. The algorithm was initially designed in the membership and justifying assignments model, and then a generic transformation was applied (MJ$\rightarrow$ME, described below) to produce an algorithm in the membership and equivalence query model. This technique does not immediately imply an interpolation result (membership queries only) for the monotone case. However, we were able to derive such an interpolation result by demonstrating another generic transformation (MJ(U)$\rightarrow$ M(M), described below) that converts a membership query and justifying assignment algorithm for learning a unate class into a membership query only interpolation algorithm for the corresponding monotone class. Unfortunately, the resulting algorithm is a factor of n less efficient than the membership query only interpolation algorithm presented in this paper (hence our decision not to present it here).

The transformations in this paper have also been used in designing algorithms for learning read-once formulas over other bases [HH 91], [BHH 91a], [BHH 91b].

We note that for some classes of formulas (including the restricted classes of read-once formulas discussed in [GKS 90a], and the arithmetic read-once formulas discussed in [BHH 91a]), justifying assignments can be generated with high probability using random membership queries. For such classes, it is trivial to transform a learning algorithm using membership queries and justifying assignments into an algorithm that interpolates (with high probability) using deterministic and random membership queries.

The first transformation we present is a very simple one that we call M(M)$\rightarrow$MJ(U), which converts a membership query algorithm for learning a monotone class into a membership query and justifying assignments algorithm for learning the corresponding unate class (This transformation is almost identical to one presented in [AHK 89]. We present it here for completeness). We then present a second transformation, MJ$\rightarrow$ME which converts an algorithm for learning a class of formulas in the membership and justifying assignments model, into an algorithm for learning the same class of formulas in the membership and equivalence query model.

The main transformation presented in [AHK 89] is M(M)$\rightarrow$ME(U) (Membership query algorithm for monotone class $\rightarrow$ Membership and equivalence query algorithm for unate class). By combining the M(M)$\rightarrow$MJ(U) transformation with the MJ$\rightarrow$ME transformation, we get an M(M)$\rightarrow$ME(U) transformation which is slightly different from the one presented in [AHK 89]. As in [AHK 89], we also show that these transformations can be modified to include equivalence queries in the base and target models of the transformation (i.e. we argue that transformations of the form ME(M)$\rightarrow$MJE(U), MJE$\rightarrow$ME, and hence ME(M)$\rightarrow$ME(U) all exist).

We then present the transformation MJ(U)$\rightarrow$M(M), which is the reverse of the first transformation. This is also shown to hold when equivalence queries are allowed, so MJE(U)$\rightarrow$ME(M) also exists.

In contrast, we show that the reverse of the ME$\rightarrow$MJ transformation does not exist in general. We present a class of formulas that can be learned in the model of membership and equivalence queries but not in the model of membership queries and justifying assignments.

Some of the results in this paper appeared in a preliminary form in [HK 91], [Han 90], and [HH 91].

2 Basic Definitions

Let V_n denote the set $\{X_1, X_2, ..., X_n\}$. An assignment A to V_n can be denoted by giving the vector $[A(X_1), \ldots, A(X_n)]$, where $A(X_i)$ is the value assigned to X_i by A.

We say that a formula f is defined on the variable set V_n if the variables in f are a subset of V_n. If A is an assignment to the variables in V_n and f is defined on V_n, then we denote by $f(A)$ the output of the formula f when its inputs are set according to the assignment A.

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 $V_n - V'$.

For $X \in V_n$, let $A_{X \leftarrow k}$ denote the assignment B such that $B(Y) = A(Y)$ for all $Y \in V_n - \{X\}$, and $B(X) = k$. Let $A_{\neg X}$ denote the assignment B such that $B(Y) = A(Y)$ for all $Y \in V_n - \{X\}$, and $B(X) = \neg A(X)$.

If f is defined on V_n, A is an assignment to V_n, $X \in V_n$, and $f(A) \neq f(A_{\neg X})$, then A is *justifying* for X in f.

Let $V' \subseteq V_n$. We say that a formula f *depends on* the variables in V' if for every $X \in V'$, there is a justifying assignment for X in f.

A *partial assignment* P to V_n can be denoted by a vector $[P(X_1), \ldots, P(X_n)]$ where each $P(X_i) \in \{0, 1, *\}$. We say that a variable X_i in V_n is *assigned by* P if $P(X_i) \neq *$. If A is an assignment to V_n, and P is a partial assignment to V_n, then we denote by P/A the assignment C to V such that $C(X_i) = P(X_i)$ for all X_i such that $P(X_i) \neq *$, and $C(X_i) = A(X_i)$ for all X_i such that $P(X_i) = *$.

If a formula f is defined on V_n, then each partial assignment P to V_n induces a *projection* f_P of f which is the formula obtained from f by replacing by the appropriate constants those variables in f to which P assigns a value.

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 functions* are functions of the form Th_k^m. Note that Th_1^m computes the OR of m variables, and Th_m^m computes the AND of m variables. Thus the boolean threshold basis includes the basis (AND,OR).

A boolean formula is *monotone* if all of its gates compute monotone functions. 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 any monotone boolean formula over V, let $U(f)$ denote the class of all formulas f' obtained from f by selecting a subset V' of V and replacing every occurrence of X_i in V' by $\neg X_i$. If M is a class of monotone boolean formulas, let $U(M)$ denote the union of $U(f)$ for all $f \in M$. All elements of $U(M)$ are unate, and we call $U(M)$ the *unate class corresponding to M.*

We define the class of *read-once threshold formulas* to be the class of read-once formulas over the basis of boolean threshold functions and negation. The class of *monotone read-once threshold formulas* consists of read-once formulas whose gates all compute functions of the form Th_k^m (no negations).

Because $\neg Th_k^m(x_1, x_2, ..., x_m) = Th_{m-k}^m(\neg x_1, \neg x_2, .., \neg x_m)$, it is possible to rewrite every read-once threshold formula so that all negations occur next to the variables. Thus every read-once threshold formula is equivalent to a unate read-once threshold formula. We will therefore assume, without loss of generality, that all read-once threshold formulas are unate. It follows that the unate class corresponding to the class of monotone read-once threshold formulas is the class of (not necessarily monotone) read-once threshold formulas.

Let f be a monotone boolean formula defined on V_n. A set of variables $S \subseteq V_n$ is a *minterm* of f if for every assignment A that assigns 1 to every variable in S we have $f(A) = 1$, and this property does not hold for any proper subset S' of S. A set $T \subseteq V_n$ of variables is a *maxterm* of f if for any assignment B that assigns 0 to all the variables in T we have $f(B) = 0$, and this property does not hold for any proper subset T' of T.

A formula can be viewed as a rooted tree whose gates are internal nodes labelled by the function computed by the gate, and whose leaves contain variables or constants. If a formula f is read-once, then for every pair of variables X and Y in f, there is a unique node farthest from the root that is an ancestor of both X and Y called their *lowest common ancestor*, which we write as $\mathrm{lca}(X, Y)$.

2.1 Identification with queries and justifying assignments

The learning criterion we consider is *exact identification*. There is a formula f called the *target formula*, which is a member of a class of formulas C defined over the variable set V_n. The goal of the learning algorithm is to halt and output a formula f from C that is logically equivalent to f.

In a *membership query*, the learning algorithm supplies an assignment A to the variables in V_n as input to a *membership oracle*, and receives in return the value of $f(A)$. Note that because $f_P(A) = f(P/A)$ it is possible to simulate a membership oracle for the projection f_P using a membership oracle for f.

In an *equivalence query*, the learning algorithm supplies a formula h from the class C as input to an *equivalence oracle*, and the reply of the oracle is either "yes", signifying that h is equivalent to f, or a *counterexample*, which is an assignment B such that $h(B) \neq f(B)$. The counterexample can be any such assignment B, and an algorithm that learns using equivalence queries is expected to perform properly no matter which counterexamples are produced.

A *set of justifying assignments* for a formula f contains, for every relevant variable X in f, a pair (X, A) such that A is a justifying assignment for X in f. When we say that an algorithm uses justifying assignments, we mean that the algorithm must be given as input a set of justifying assignments for the target function f.

3 Learning Monotone Read-Once Threshold Formulas

We present an algorithm for exactly learning monotone read-once threshold formulas in polynomial time using membership queries.

3.1 Findmin

Our algorithm for learning monotone read-once threshold formulas 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 Q containing a maxterm of f, and outputs a maxterm contained in Q.

3.2 The basic subroutine

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

We defer the presentation of these 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 3.6. The algorithm is recursive, and it learns the target formula f depth first.

We assume without loss of generality that the target formula f contains no constants in its leaves, and that it is *non-degenerate*, in that there are no adjacent AND gates or adjacent OR gates along a root leaf path. Any monotone read-once threshold formula can be rewritten to satisfy these conditions.

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 $\mathit{LcaRootT}^f$ to find the set T' of variables Y in $T - \{X\}$ such that $\mathit{lca}(X, Y)$ 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 $\mathit{LcaRootS}^f$ 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 outputting the formula $Th_k^m(f_1, f_2, \ldots, f_m)$.

3.4 Lemmas

The algorithm is based on the following lemmas (and their duals).

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. $\square$

Lemma 2 *Let f be a monotone read-once threshold 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 $Z \subseteq V'$. Let S be the minterm of f output by Findmin$^f(V')$. 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 $S \cap Z \neq \emptyset$.

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 $S \cap Z \neq \emptyset$. 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 $\mathit{Findmin}^f(V')$, S' contains a set which is a superset of S. S contains a minterm of g, and therefore $g(1_S) = 1$. 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 monotone read-once threshold formula defined on the variable set V_n. Let T be a maxterm of f. Let S be the output of $\mathit{Findmin}^f(V')$ for some $V' \supseteq V_n - T$ and let $S \cap T = \{X\}$. If $Y \in T - \{X\}$ and S_Y is the minterm output by $\mathit{Findmin}^f((V_n - T) \cup \{Y\})$, 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 G be a gate which is on the path from $lca(X,Y)$ to the root such that G is not equal to $lca(X,Y)$. S and S_Y contain minterms of the same subformulas (rooted at children of G). Since T contains no variables from these subformulas, V' and $(V_n - T) \cup \{Y\}$ will include the all variables from such subformulas. By Lemma 2, S and S_Y 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. $\Box$

Lemma 4 *Let f be a monotone read-once threshold formula defined on the variable set V_n. Let T be a maxterm of f. Let S be the output of $Findmin^f(V')$ for some $V' \supseteq V_n - T$ and 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

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

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 G be the gate which is the child of the root, on the path from X to the root. The gate G is also on the path from Y to the root. Since f is a (non-degenerate) monotone read-once threshold formula, G is not an OR gate. Therefore, there exists a subformula h rooted at a child of G such that T does not contain a maxterm of h. Clearly V' and $(V_n - T) \cup \{Y\}$ both contain all the variables of h. Since $S \cap T = \{X\}$, S must contain minterms of all the subformulas of G for which T does not contain a maxterm, and hence S contains a minterm of h. Similarly, S_Y contains a minterm of h. By Lemma 2, S and S_Y will contain the same minterm of h, and therefore $S \cap S_Y$ is not empty.

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

Suppose the root is Th^m_k ($k \neq 1$). 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 $V' \supseteq V_n - T$.

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 $T - \{X\}$ $\qquad S_Y := \mathit{Findmin}^f((V_n - T) \cup \{Y\})$.

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.

3.6 The Algorithm

$$\mathit{MROTLearn}^f$$

1. $S := \mathit{Findmin}^f(V_n)$

2. Pick an X in S.
   $T := \mathit{Findmax}^f((V_n - S) \cup \{X\})$

3. if $S = T = \{X\}$, then return(X) (the formula f is equal to X)

4. $T' := \mathit{LcaRootT}^f(S, T, X)$

```
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
       $MROTLearn^{f_1}$, 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) $T_{X'} := Findmax^f((V_n - S) \cup \{X'\})$
   (c) $S' := LcaRootS^f(S, T_{X'}, X')$
   (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
       $MROTLearn^{f_k}$, 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) $S_{X'} := Findmin^f((V_n - T) \cup \{X'\})$
```

(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 $\mathit{MROTLearn}^{f_{k+j-1}}$, simulating calls to the membership oracle of f_{k+j-1} with calls to the membership oracle of f.

9. Output the formula $Th_k^{k+j-1}(f_1, f_2, f_3, \cdots, f_{k+j-1})$

4 Correctness and Complexity

Theorem 1 *There is a learning algorithm that exactly identifies any monotone read-once threshold 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 four lemmas (and their duals) 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 MROT^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 MROT^{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 $MROTLearn^{f_k}$ 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 $MROT^f$, 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. $\square$

Corollary 1.1 *There is a learning algorithm that exactly identifies any read-once threshold formula in time $O(n^4)$ using $O(n^4)$ membership queries and $O(n)$ equivalence queries.*

Proof: The class of read-once threshold formulas is the unate extension of the class of monotone read-once threshold 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. $\square$

5 Transformations

5.1 M(M)→MJ(U)

Let f be a unate formula. Since f is unate, all negations in f appear at the leaves. If X is a variable in f and X is negated in f (i.e. there is a negation appearing next to all occurrences of X in f) then we say the sign of X is negative in f. Otherwise, we say the sign of X is positive. If the sign of X is negative, then (because f is unate) for all assignments A, $f(A_{X\leftarrow 0}) \geq f(A_{X\leftarrow 1})$. If the sign of X is positive, then for all assignments A, $f(A_{X\leftarrow 0}) \leq f(A_{X\leftarrow 1})$.

Suppose you are given a justifying assignment A for a variable X in f. Because A is justifying for X, the values of f on $A_{X\leftarrow 0}$ and $A_{X\leftarrow 1}$ are distinct. Therefore, if $A_{X\leftarrow 0} = 1$ then the sign of X is negative, and if $A_{X\leftarrow 0} = 0$ then the sign of X is positive. It follows that to find the sign of a variable X in f, given a justifying assignment A for X in f, it suffices to ask the membership query $A_{X\leftarrow 0}$.

Given a unate formula f, define a corresponding monotone formula f' that is derived from f as follows: for every X_i in f that is negative, replace all occurrences of $\neg X_i$ in f with a new variable Y_i. The formula f' is a monotone formula that is closely related to f. If we know the signs of the variables in f, it is easy to simulate a membership oracle for f' using a membership oracle for f. To answer the membership query "What is $f'(A')$?" using a membership oracle for f, we simply create an assignment A such that $A[X_i] = A'[X_i]$ if the sign of X is positive, and $A[Y_i] = \neg A'[X_i]$ if the sign of X is negative. Since $f'(A') = f(A)$ to discover $f'(A')$, it suffices to query the membership oracle for f on assignment A.

The above observations give a simple method of transforming an algorithm for learning a monotone class of formulas using membership queries into an algorithm for learning the corresponding unate class using membership queries and justifying assignments. Let $AlgMM$ be an algorithm for learning a class of monotone formulas using membership queries. Let S be a set of justifying assignments for the variables in f. Discover the signs of the variables in f using the justifying assignments in S. Let f' be the monotone formula corresponding to f as above. Use AlgMM to learn f', simulating a membership oracle for f' using the membership oracle for f. When AlgMM outputs f' (or an equivalent formula), replace each new variable Y_i with the literal $\neg X_i$, yielding a formula equivalent to f. Output this formula.

Note that the transformation can be modified to handle equivalence queries in the base model, i.e. to show ME(M)→MJE(U). The only difference is that in simulating the algorithm from the base model to learn f', we must simulate an equivalence oracle for f' using the equivalence oracle for f. This can be done simply as follows. If the question is "Does $g' = f'$?" (where g' is a formula over the variables on which f' is defined), then we convert g' into a formula g produced by replacing each variable Y_i with the literal $\neg X_i$, and asking the equivalence oracle for f, "Does $g = f$". If the answer is yes, then we've learned f and we halt and output it. If the answer is a counterexample A, then we continue the simulation by answering no to the question "Does $g' = f'$?" and returning the counterexample A' such that $A'[X_i] = A[X_i]$ for all variables X_i with positive signs in f, and $A'[Y_i] = \neg A[X_i]$ for all variables X_i with negative signs in f.

The above transformations give the following theorem.

Theorem 2 *Let M be a monotone class of formulas. If M can be exactly identified in polynomial time by an algorithm using membership queries then $U(M)$ can be exactly identified in polynomial time by an algorithm using membership queries and* *justifying assignments. Furthermore, if M can be exactly identified in polynomial time by an algorithm using membership and equivalence queries, then $U(M)$ can be exactly identified in polynomial time by an algorithm using membership queries, equivalence queries, and justifying assignments.*

5.2 MJ→ME

Let AlgMJ be an algorithm using membership queries and justifying assignments. The transformation algorithm, ToME(AlgMJ), which uses membership and equivalence queries, is based on a loop. At the start of the loop, we have a subset V' of V, and a projection p assigning values to the variables in $V_n - V'$ and leaving the variables in V' unassigned. For every $X \in V'$, we know a justifying assignment for X in f_P. For the first iteration of the loop, we set V' to empty, and we set p to an arbitrary assignment to V_n, so f_P is constant.

We run AlgMJ to learn f_P (simulating a membership oracle for f_P with the membership oracle for f). The output of AlgMJ is a formula $g \equiv f_P$. We ask the equivalence query "$g \equiv f$?" (i.e. $f_P \equiv f$?). If the answer is "yes" (which it will be when V' contains all the relevant variables of f), we are done. If the answer is "no" the equivalence oracle returns a counterexample A. We call a routine *ProjBitFlip* (described below) that returns a new, larger subset V' of V_n, a new associated projection P, and justifying assignments (with respect to f_P) for the variables in the new V'. We then repeat the loop. Termination of ToME(AlgMJ) is guaranteed by the fact that at each iteration we increase the size of V'.

ProjBitFlip takes as input V', the projection P, the counterexample A (and assignment on which f_P differs from f), and a set S of justifying assignments for the variables in V' (the assignments in S are justifying with respect to f and f_P). The key processing in ProjBitFlip is a loop to greedily reduce the number of variables on which A and P differ.

ProjBitFlip(V',P,A, S)

1. Set B to A. Set W to be the set of variables in $V_n - V'$ such that $B(X) \neq P(X)$.
2. While there exists an $X \in W$ such that $f(B_{\neg X}) = f(b)$, set B to $B_{\neg X}$ and delete X from W.
3. Let P' be the projection such that $P'(X) = P(X)$ for all $X \in V_n - W$, and $P'(X) = *$ for all $X \in W$.
4. Let $S' = S \bigcup \{(X, B) | X \in W\}$.
5. Output $V' \bigcup W$, P', and S'.

The processing in ProjBitFlip does not change either $f(B)$ or $f_P(B)$, so for the final B those two values still differ. This means $B \neq P/B$ and therefore W is not empty. Our new variable set is $V' \bigcup W$, and P' assigns $*$ to all variables in W. Assignment B is justifying for those new variables in $f_{P'}$ and in f.

For completeness, we present the transformation algorithm ToME(AlgMJ).

ToME(AlgMJ)

1. Let $W = \emptyset$. Let $S = \emptyset$.
2. Do forever:
    (a) Let P be an arbitrary partial assignment whose defined set is $V_n - V'$.

(b) Call the procedure AlgMJ using the justifying assignments S, and simulating membership queries to the function f_P. Let g be the formula returned.

(c) Make an equivalence query with g. If the reply is "yes" then output g and halt, otherwise, let A be the counterexample.

(d) In this case, $g(A) = f_P(A) \neq f(A)$, that is, $f(P/A) \neq f(A)$. Call ProjBitFlip(V', P, A, S). Set V', P and S to the values W, P' and S' returned by ProjBitFlip.

The techniques in [AHK] show that a modified version of ToME works if the input algorithm uses equivalence queries as well as membership queries and justifying assignments. The modification is mainly to step 2b, where we are simulating AlgMJ to learn f_P, even though we only have oracles for f. If AlgMJ asks an equivalence query "Does $g = f_P$?", we can do the following:

- Ask the equivalence oracle for f, "Does $g = f$?" If the reply is "yes", then halt and output g. Otherwise, the reply is no and a counterexample A. Ask the membership oracle for f for the value of $f(P/A)$ (which equals $f_P(A)$). Evaluate $g(A)$. If $g(A) \neq f_P(A)$, then A is a counterexample to the query asked by AlgMJ "Does $g = f_P$?", so continue the simulation by returning the counterexample A. If $g(A) = f_P(A)$, then $f_P(A) \neq f(A)$. Discontinue the simulation, and jump to step 2d (because we already have an assignment A such that $f_P(A) \neq f(A)$, which is what we need in step 2d to call ProjBitFlip and expand the set V').

We have the following theorem.

Theorem 3 *If C is a projection closed class of formulas such that C can be exactly identified in polynomial time by an algorithm using membership queries, equivalence queries, and justifying assignments, then C can be exactly identified in polynomial time by an algorithm using membership and equivalence queries.*

5.3 ME$\rightarrow$MJ does not exist

We present a class of formulas for which there exists a polynomial time membership and equivalence query learning algorithm, but for which no polynomial time membership query and justifying assignment algorithm exists.

Consider the subclass of monotone DNF formulas defined over the variable set $V_n = \{x_1, ..., x_n\}$ where $n = m^2$, consisting of $m + 1$ terms of the following type. The first m terms are $x_1 \bigwedge x_2 \bigwedge \cdots \bigwedge x_m$, $x_{m+1} \bigwedge x_{m+2} \bigwedge \cdots \bigwedge x_{2m}$, $\ldots$, $x_{(m-1)m+1} \bigwedge x_{(m-1)m+2} \bigwedge \cdots \bigwedge x_n$. The last term is made up of all but m variables, with one variable missing from each of the first m terms. This class is learnable by the membership and equivalence query algorithm for learning monotone DNF [A 87].

The formulas in this class differ only in their last term, so learning a formula in this class is equivalent to learning the last term in the formula. Note that for a given formula in the class, there is exactly one assignment that sets only the last term to true – the assignment in which all variables in the last term are set to 1, and all other variables are set to 0. There are m^m possibilities for this last term.

Let AlgMJ be an algorithm for learning this class with membership queries and justifying assignments. No matter what the target formula, the algorithm could be given the same set of justifying assignments – the assignment setting $x_1 ... x_m$ to 1 and the other variables to 0 is justifying for the variables $x_1 ... x_m$, the assignment setting $x_{m+1}...x_{2m}$ to 1 and the other variables to 0 is justifying for the variables $x_{m+1}...x_{2m}$ and so forth. Suppose AlgMJ is given this set of justifying assignments. Consider the following adversary strategy. Each time AlgMJ asks a membership query which sets one of the first (known) m terms to 1, answer 1. Each time AlgMJ asks a membership query which doesn't set any of the first (known) m terms to 1, answer 0. Since for each possible final term of the target formula, there is exactly one assignment which sets this term to 1 without setting one of the first m terms to 1, it follows that each time the adversary answers 0 it eliminates at most 1 candidate for the last term of the target formula. Thus to uniquely identify the last term of the target formula (and learn the target formula) takes at least $m^m - 1$ membership queries, given this set of justifying assignments. It follows that there is no polynomial time algorithm for learning this class using membership queries and justifying assignments.

5.4 MJ(U)$\rightarrow$M(M)

To perform this transformation, we need to show how we can generate justifying assignments for the variables in a monotone class using membership queries.

The transformation is based on the following observation. Suppose f is a monotone formula defined on the variable set V_n. Let V' be a subset of the variables in V_n. Let f_{P_0} be the projection setting the variables in V' to 0 and leaving the variables in $V_n - V'$ unassigned. Let f_{P_1} be the projection setting the variables in V' to 1 and leaving the other variables in $V_n - V'$ unassigned. Suppose there is a relevant variable X in $V_n - V'$. Let A be a justifying assignment for that variable such that $f(A) = 0$. Then, because f is monotone, $A(X) = 0$ and $f(A_{X\leftarrow 1}) = 1$. It also follows from the monotonicity of f that $f_{P_0}(A) = f(P_0/A) \leq f(A)$ and

$f_{P_1}(A) = f(P_1/A) \geq f(A)$. Therefore, $f_{P_0}(A) = 0$ and $f_{P_1}(A) = 1$ and $f_{P_0} \neq f_{P_1}$. In summary, if $V_n - V$ contains a relevant variable of f, then $f_{P_0} \neq f_{P_1}$. In contrast, it is clear that if $V_n - V$ does not contain any relevant variables of f, then $f_{P_0} = f_{P_1}$.

Let AlgMJU be an algorithm for learning a unate class C' of formulas using membership queries and justifying assignments. We present a transformation algorithm ToMM(AlgMJU) which learns the corresponding monotone class C of formulas using only membership queries.

The transformation algorithm is based on a loop. At the start of the loop, we have a subset V' of V and a set S of justifying assignments for the variables in V' (the assignments are justifying with respect to f). With V' we associate the projections f_{P_0} and f_{P_1}, which are the projections obtained by setting the variables in $V - V'$ all to 0 and all to 1, respectively. We check to see whether for all (X, A) in S, A is also justifying for X in f_{P_0} and f_{P_1}. If this is not the case, then we will expand V' and S as follows. Note that because A is justifying for X in f, $f(A_{X\leftarrow 0}) = 0$ and $f(A_{X\leftarrow 1}) = 1$, and hence by monotonicity $f_{P_0}(A_{X\leftarrow 0}) = 0$ and $f_{P_1}(A_{X\leftarrow 1}) = 1$. It follows that if A is not justifying for X in both f_{P_0} and f_{P_1}, then either $f_{P_1}(A_{X\leftarrow 0}) \neq f_{P_0}(A_{X\leftarrow 0})$ or $f_{P_0}(A_{X\leftarrow 1}) \neq f_{P_1}(A_{X\leftarrow 1})$. Suppose that $f_{P_1}(A_{X\leftarrow 0}) \neq f_{P_0}(A_{X\leftarrow 0})$ (the other case is similar). Then $f_{P_1}(A_{X\leftarrow 0}/P_0) = f_{P_1}(A_{X\leftarrow 0}) \neq f(A_{X\leftarrow 0}/P_0)$, meaning that f_{P_1} and f differ on assignment $A_{X\leftarrow 0}/P_0$. We exploit the assignment $A_{X\leftarrow 0}/P_0$ (calling ProjBitFlip) to find a justifying assignment (with respect to f) for a variable Y in $V - V'$. We then add Y to V', add Y and its justifying assignment to S, and go back to the start of the loop.

When we reach the point where all the assignments in S are justifying for both f_{P_0} and f_{P_1}, we run two parallel simulations of AlgMJU with the set S of justifying assignments as input. In one simulation we answer a membership queries by simulating a membership oracle for f_{P_0}, and in the other simulation we answer membership queries by simulating a membership oracle for f_{P_1}. If the simulations terminate without ever diverging (doing anything different) then the two simulations will output the same formula g, meaning that $g = f_{P_1} = f_{P_0} = f$. In this case we halt and output g. If the two simulations diverge, then they do so because at some point the answer to a membership query on an assignment B is answered differently in the two simulations, and thus $f_{P_1}(B) \neq f_{P_0}(B)$, and so $f_{P_1}(B/P_0) \neq f(B/P_0)$. In this case we exploit the assignment B/P_0 (calling ProjBitFlip) to find a justifying assignment for a variable Y in $V - V'$. We then add Y to V', add Y and its justifying assignment to S, and go back to the start of the loop.

We present the transformed algorithm below.

ToMM(AlgMJU)

1. Let $W = \emptyset$. Let $S = \emptyset$.
2. Do forever:
   (a) Let P_0 be the partial assignment setting the variables in $V - V'$ to 0 and leaving the variables V' unassigned. Let P_1 be the partial assignment setting the variables in V' to 1 and leaving the variables in V' unassigned.
   (b) If there is a pair $(X, A) \in S$ such that A is not a justifying assignment for X in f_{P_0}, then call ProjBitFlip(V',P_1,$A_{X \leftarrow 0}/P_0$, S) and set V' and S respectively to the values W, and S' returned.
   (c) else if there is a pair $(X, A) \in S$ such that A is not a justifying assignment for X in f_{P_1}, then call ProjBitFlip(V',P_0,$A_{X \leftarrow 1}/P_1$, S) and set V' and S respectively to the values W and S' returned.

(d) else run two parallel simulations of AlgMJU on input S to learn f_{P_0} and f_{P_1}. If the two simulations diverge on some membership query B then call ProjBitFlip(V',P_1,B/P_0, S) and set V' and S respectively to the values W and S' else if the two simulations do not diverge and they both output the same formula g, halt and output g.

We can also modify this MJ(U)→M(M) transformation to include equivalence queries in the base and target models, i.e. to produce the transformation MJE(U)→ME(M). The modification is basically the same as that described at the end of Section 5.2. Thus we have the following theorem.

Theorem 4 *Let $U(M)$ be a unate, projection closed class of formulas corresponding to a monotone class M. If $U(M)$ can be exactly identified in polynomial time by an algorithm using membership queries, and justifying assignments then M can be exactly identified in polynomial time by an algorithm using only membership queries. Furthermore, if $U(M)$ can be exactly identified in polynomial time by an algorithm using membership queries, equivalence queries, and justifying assignments, then M can be identified in polynomial time using only membership and equivalence queries.*

References

[A 87] D. Angluin. Queries and concept learning. In *Machine Learning*, 2:319–342, 1987.

[AHK 89] D. Angluin, L. Hellerstein, and M. Karpinski. Learning read-once formulas with queries. Technical report, Report No. UCB/CSD 89/528, Computer Science Division, University of California Berkeley, 1989. To appear, *J. ACM* '91.

[BHH 91a] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning arithmetic read-once formulas; to appear in 24^{th} ACM STOC 1992.

[BHH 91b] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning boolean read-once formulas over extended bases. Manuscript in Preparation.

[GKS 90a] Sally A. Goldman, Michael J. Kearns, and Robert E. Schapire. Exact identification of circuits using fixed points of amplification functions. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, 1990.

[Han 90] Thomas Hancock. Identifying μ-formula decision trees with queries. Technical report, Harvard University TR-16-90, 1990.

[HH 91] Thomas Hancock and Lisa Hellerstein. Learning read-once formulas over fields and extended bases. In *The 1991 Workshop on Computational Learning Theory*, 1991.

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

[HK 91] L. Hellerstein and M. Karpinski. Computational Complexity of Learning Read-Once Formulas over Different Bases Technical Report, International Computer Science Institute TR-91-014, 1991.