Journal of the Association for Computing machinery 40:1, 1993.

Learning Read-Once Formulas with Queries

Dana Angluin * Lisa Hellerstein † Marek Karpinski ‡

Abstract

A read-once formula is a boolean formula in which each variable occurs at most once. Such formulas are also called μ-formulas or boolean trees. This paper treats the problem of exactly identifying an unknown read-once formula using specific kinds of queries.

The main results are a polynomial time algorithm for exact identification of monotone read-once formulas using only membership queries, and a polynomial time algorithm for exact identification of general read-once formulas using equivalence and membership queries (a protocol based on the notion of a *minimally adequate teacher* [1]). Our results improve on Valiant's previous results for read-once formulas [26]. We also show that no polynomial time algorithm using only membership queries or only equivalence queries can exactly identify all read-once formulas.

1 Introduction

The goal of computational learning theory is to define and study useful models of learning phenomena from an algorithmic point of view. Since there are a variety of real-world learning problems, differing in the kind and quality of information available to the learner and the performance requirements on the learner, one would expect a corresponding variety of useful models. For example, it seems unreasonable to expect a single definition to model learning well at both the neuron level and higher cognitive levels.

Stimulated by Valiant's seminal paper [26], much recent research has focused on exploring definitions of "efficient learnability." Valiant's paper addressed the question of which syntactically defined classes of boolean formulas can be efficiently learned.

*Computer Science Dept., Yale University, P. O. Box 2158, New Haven, CT 06520. Research supported by NSF grant IRI-8718975.

†Computer Science Division University of California Berkeley, CA 94720. Research supported by NSF Grant CCR-8411954, and an AT&T GRPW grant.

‡International Computer Science Institute Berkeley, CA 94704. On leave from the University of Bonn. Research partially supported by DFG Grant KA 673/2-1, and by SERC Grant GR-E 68297.

The primary criterion of efficient learnability introduced by Valiant was that of polynomial time distribution-free approximate identification of concepts from randomly chosen correctly classified examples. This criterion subsequently received a great deal of attention. However, Valiant also considered other sources of information about the unknown concept, which he termed "oracles." He defined three specific types of oracle: "necessity", "relevant possibility", and "relevant accompaniment".

Valiant showed that monotone disjunctive normal form formulas can be approximately identified in polynomial time from random examples and the necessity oracle. He also showed that general read-once formulas can be identified by a polynomial time algorithm using all three of these oracles and no random examples. In this case, the identification is "exact", that is, the final formula is exactly rather than approximately equivalent to the unknown formula.

Read-once formulas are a natural special case to consider in studying the learnability of boolean formulas. Any boolean formula can be efficiently represented as an AND/OR tree with its leaves labelled by literals, but in the case of a read-once formula, no two leaves are labelled by literals of the same variable. This means that if we start with an assignment of truth values to the variables of a read-once formula, and change the value assigned to one variable X_i, this change will only affect the values computed by nodes along the path from the root to the (at most one) leaf containing an occurrence of X_i. This contrasts with the case of a general boolean formula, in which several paths may be simultaneously affected by a single change of this type.

Given this initial intuition, the following negative results are surprising. Pitt and Valiant [22] show that if RP $\neq$ NP, then there is no polynomial time algorithm to learn even monotone read-once formulas from examples in the distribution free model. In this case, the restriction that the hypothesis of the learning algorithm be represented as a read-once formula is essential to the proof.

If instead we consider the problem of predicting the value of the formula on randomly chosen examples (see [23] for definitions), a potentially easier problem, the reductions given by Kearns, Li, Pitt, and Valiant [19] show that monotone read-once formulas are no easier to predict in the distribution free model than general boolean formulas. Kearns and Valiant [20, 18] have shown that prediction of general boolean formulas in the distribution free model is as hard as certain cryptographic problems, for example, factoring Blum integers. Thus, it seems that we must move away from the distribution free model in order to exploit the special properties of read-once formulas.

Angluin [2] has proposed studying the notion of polynomial time exact identifiability of concepts using various types of queries (or oracles, in Valiant's terminology.) Two types of queries seem to be particularly interesting: membership queries and equivalence queries.

In a membership query, the learning algorithm proposes a particular example, and the reply is a correct classification of the example according to the unknown concept. In an equivalence query, the learning algorithm proposes a hypothesis in a specified hypothesis language, and the reply is either "yes" or a counterexample. The answer "yes" signifies that the hypothesis is equivalent to the unknown concept. A counterexample is a particular example that is classified differently by the unknown concept and the proposed hypothesis, and thus is a witness of their inequivalence. The choice of which counterexample to present in response to a given equivalence query is assumed to be arbitrary – a successful learning algorithm must work no matter which one is chosen.

To illustrate these queries, imagine a student attempting to learn the concept of "the main verb in an English sentence." In attempting to grasp this concept, the student may ask the teacher a question of the form "Is 'enter' the main verb of 'Abandon hope, all ye who enter here'?" This is a membership query, and the correct answer is "no".

However, an equivalence query seems a bit suspect here – the student is required to produce a complete description of his or her current hypothesis about the concept of "the main verb in an English sentence," and then the teacher is required to determine whether it is correct or not, and come up with a counterexample if not. In fact, it seems downright hopeless.

The example itself suggests a more practical alternative – the teacher can test the student by giving a collection of examples and requiring the student to classify whether they are instances of the concept of "the main verb in an English sentence." If the student misclassifies any of the examples, then this example serves as a counterexample to the (never explicitly stated) hypothesis of the student about the concept. If the student classifies enough examples correctly, the teacher may be content to assume that the student's hypothesis is "sufficiently correct."

One possible formal analog of this is to use the distribution free model of Valiant to supply the "testing" portion of this example. In particular, a learning algorithm that uses equivalence queries and membership queries and achieves exact identification may be transformed into one that uses randomly drawn examples and membership queries and achieves approximate identification in the distribution free model, with a polynomial increase in computational cost [1, 2]. The idea is to replace each equivalence query with a sufficient quantity of randomly drawn examples, checking that the hypothesis classifies each one correctly. If so, the equivalence query is answered "yes" and with high probability the hypothesis is approximately equal to the unknown concept. If not, the equivalence query is answered with the incorrectly classified example as a counterexample.

Littlestone [21] demonstrates another possible efficient transformation of a learning algorithm that uses equivalence queries and achieves exact identification. If the setting is to predict the classification of each of a sequence of examples, receiving the correct classification after each prediction, then a learning algorithm that uses equivalence queries and achieves exact identification can be transformed into a prediction algorithm that in the worst case makes no more errors of prediction than it would have made equivalence queries. In this case, instead of an equivalence query, the current hypothesis is used to make predictions. As long as it predicts correctly, the hypothesis is retained. If it makes an incorrect prediction, the misclassified example serves as a counterexample to the current hypothesis. This case clearly could admit membership queries with appropriate definitions.

Thus a polynomial time exact identification algorithm that uses equivalence and membership queries can be transformed into either of these two types of more "practical" algorithms. And in these cases, the specter of the difficulty of the learner specifying and the teacher verifying a complete and formal hypothesis is dispelled.

This paper considers the complexity of exact identification of both read-once formulas and monotone read-once formulas. We note that general (i.e. not necessarily monotone) read-once formulas cannot be exactly identified in polynomial time with membership queries alone. We give a polynomial time algorithm for exact identification of *monotone* read-once formulas using membership queries. We prove that equivalence queries alone are not sufficient to exactly identify general or monotone read-once formulas. More precisely, we prove that there is no polynomial time algorithm for learning monotone read-once formulas with equivalence queries alone, even if the algorithm is allowed to use non-monotone read-once formulas in its equivalence queries. We give a polynomial time algorithm for exact identification of general read-once formulas that uses both membership and equivalence queries. We also present a polynomial time algorithm for exact identification of general read-once formulas using a relevant possibility oracle. The algorithm is an improvement of Valiant's polynomial time algorithm because it uses only one of the three oracles that Valiant's uses. Some of the results in this paper appeared in a preliminary form in [3] and [15]. Angluin [3] and Hellerstein and Karpinski [15] independently discovered polynomial time algorithms to exactly identify monotone read-once formulas using membership queries.

The problem of exact identification using only membership queries can be viewed as a problem of interpolation using a "black box" input oracle. Thus, the algorithm that we present for exact identification of monotone read-once formulas using only membership queries can also be viewed as an efficient interpolation algorithm for this class of boolean formulas from a (black box) input oracle. Corresponding results for interpolation of boolean polynomials (formulas over the basis of AND and XOR) over small extensions of finite fields are proved by Grigoriev, Karpinski, and Singer [9].

Several learning problems have been shown to have polynomial time exact identification algorithms using equivalence and membership queries. Some examples are deterministic finite state acceptors [1], monotone DNF formulas [2], bracketted context free languages described by deterministic bottom-up tree automata [25], and propositional Horn sentences [5]. In each of these cases there is a proof that no polynomial time learning algorithm is possible that uses only membership or only equivalence queries [4, 2]. To this list we now add read-once formulas.

2 Preliminaries

2.1 Boolean functions and formulas

Let n be a positive integer. The *boolean n-vectors* is the set B_n of vectors $\{0,1\}^n$. The i^{th} component of the boolean vector x is denoted $x[i]$. For two boolean vectors x and y from B_n, we define $x \leq y$ if and only if for each $i = 1, \ldots, n$, $x[i] \leq y[i]$.

A *boolean function* of n arguments is any function from B_n to $\{0,1\}$. A boolean function f of n arguments is *monotone* if and only if for any x and y from B_n, if $x \leq y$ then $f(x) \leq f(y)$. Boolean functions will be represented using boolean formulas.

The variable set V_n is defined to be $\{X_1, X_2, \ldots, X_n\}$. A *literal* is a variable X_i or its negation $\neg X_i$. X_i is a *positive* literal, $\neg X_i$ is a *negative* literal. The set of literals over V_n is denoted L_n. A subset L of L_n is *consistent* if and only if it does not contain any variable and its negation.

Boolean formulas over the variable set V_n considered here are assumed to be over the basis $\wedge$, $\vee$, and $\neg$, denoting boolean AND, OR, and NOT, respectively. The constant function 1 is denoted $\top$ and the constant function 0 is denoted $\perp$. The *size* of a boolean formula is the number of occurrences of variables that it contains. A boolean formula can be viewed as a boolean circuit in which each gate has fan-out one. The boolean formulas considered in this paper are assumed to be in a standard form in which the gates have arbitrary fan-in, the AND and OR gates are arranged in alternating levels, and negations only occur next to the variables. Every boolean formula can be transformed into this standard form by combining gates, and by applying De Morgan's Laws. A boolean formula in this form can be represented as a rooted tree whose internal vertices have at least two children and are labelled with AND or OR (in alternating levels), and whose leaves are labelled with literals.

A vector $x \in B_n$ is interpreted as an assignment to the variables in V_n, and assigns a value of 0 or 1 to every boolean formula over V_n in the usual way. 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 $V - V_n$.

A boolean formula is *monotone* if it does not contain any occurrence of $\neg$. A monotone boolean formula represents a monotone boolean function, and every monotone boolean function can be represented by a monotone boolean formula.

A boolean formula is *read-once* if and only if it contains at most one occurrence of each variable. Read-once formulas are also called *μ-formulas* or *boolean trees*. Note that the size of a read-once formula over V_n is at most n. A boolean function is *read-once* if and only if it can be represented by a read-once formula. One can show that the read-once formula representing a read-once function is unique up to tree isomorphism.

Let f be a boolean function of n arguments. A consistent set of literals $S \subset L_n$ is a *minterm* of f if for every vector x that assigns 1 to every literal in S we have $f(x) = 1$, and this property does not hold for any proper subset S' of S. A consistent set T of literals is a *maxterm* of f if for any assignment y that assigns 0 to all the literals in T we have $f(y) = 0$, and this property does not hold for any proper subset T' of T (Note: Other definitions of *minterm* and *maxterm* are sometimes used in the literature).

2.2 Identification with queries

The learning criterion we consider is that of *exact identification*. There is a formula f called the *target formula*. The formula f is a member of a class of formulas C (e.g. read-once formulas) defined over the variable set V_n. The goal of a learning algorithm for the class C is to halt and output a formula g from C that is logically equivalent to f. The learning algorithm is started with the value of n and may gather information about f by means of various types of queries.

In a *membership query*, the learning algorithm supplies a boolean n-vector x and receives in return the value of $f(x)$.

In an *equivalence query*, the learning algorithm supplies a formula h from the class C and the reply is either "yes", signifying that h is equivalent to f, or a *counterexample*, consisting of a boolean n-vector x such that $h(x) \neq f(x)$. The choice of which counterexample to supply when h is not equivalent to f is assumed to be arbitrary. In particular, the learning algorithm must work as advertised no matter which choices of counterexamples are made.

In a *relevant possibility query*, first defined by Valiant [26], the learning algorithm specifies a set S' of literals from L_n and the reply is "yes" if S' is a subset of a minterm of f and "no" otherwise.

2.3 Model of computation

Our model of computation is the random access computer (RAC) [7] augmented to allow for queries. We describe the augmentation in some detail, since we want to reflect accurately the set-up time for queries in our bounds. A random access computer is essentially a unit-cost random access machine in which each register is of length $k \log n$ bits for some constant k. The model is polynomially related to log-cost RAMs. Running times will be bounded as a function of n, the number of possible variables, which is given as an initial input to a learning algorithm.

In order to incorporate queries into this model of computation, we introduce a new unit cost instruction for each permitted type of query. In particular, for membership queries we introduce an instruction that specifies two registers W_i and W_j. The n consecutive registers beginning with W_i are each required to contain either 0 or 1 and are interpreted as representing a boolean vector x of length n. The result of the instruction is to place in the register W_j the value of $f(x)$ for the target function f.

For an equivalence query we introduce another type of instruction that specifies three registers W_i, W_j, and W_k. The contents of W_i must be a positive integer l, and the l consecutive registers starting with register W_j are taken as representing a formula g in some reasonable encoding with the property that $O(n)$ registers are sufficient to encode any read-once formula over V_n. The result of the instruction is to place in W_k the value 2 if g is equivalent to the target function f. Otherwise, the n consecutive registers beginning with W_k are assigned 0's and 1's representing a counterexample vector x.

For a relevant possibility query, we introduce a third type of instruction, which specifies three registers W_i, W_j, and W_k. W_i is required to contain a positive integer l. The l consecutive pairs of registers beginning with W_j are interpreted as a list of l literals, where the first register in the pair gives the sign of the literal, and the second gives the index of the variable in V_n. The result of the instruction is to place a 1 in register W_k if the specified set of literals is a subset of a minterm of f, and a 0 in W_k otherwise.

2.4 Properties of read-once formulas

Let f be a non-constant read-once formula. Consider the tree that represents f. If v is a vertex of the tree, $l(v)$ denotes its label, either AND, OR or a literal. Since f is read-once, no two leaves are labelled with literals of the same variable. We sometimes use the label of a leaf to denote the leaf itself.

For each vertex of the tree there is a unique path to the root. An *ancestor* of a vertex v is any vertex w on the path between v and the root, including v itself. A *descendant* of v is any vertex of which v is an ancestor. For any pair of vertices v and w in the tree, there is a unique vertex farthest from the root that is an ancestor of both v and w, called their *lowest common ancestor* and denoted $lca(v, w)$. If v and w are distinct vertices in the tree then $lca(v, w)$ is an internal vertex and $l(lca(v, w))$ is either AND or OR.

If f is a nonconstant read-once formula that is the AND of subformulas $f_1, \ldots, f_k$, then S is a minterm of f if and only if it is the union of $S_1, \ldots, S_k$, where S_i is a minterm of f_i for each i. T is maxterm of f if and only if it is a maxterm of f_i for exactly one value of i.

Similarly, if f is a nonconstant read-once formula that is the OR of subformulas $f_1, \ldots, f_k$, then T is a maxterm of f if and only if it is the union of $T_1, \ldots, T_k$, where T_i is a maxterm of f_i for each i. S is minterm of f if and only if it is a minterm of f_i for exactly one value of i.

The following pair of lemmas characterize the subsets of minterms and maxterms of f in terms of the labels of the pairwise lowest common ancestors of the elements of the subset.

Lemma 1 *Let f be a nonconstant read-once formula, and let S' be a set of literals. Then S' is a subset of a minterm of f if and only if every literal in S' occurs in f and for every pair of distinct literals Y and Z in S', $l(lca(X, Y)) =$ AND.*

Lemma 2 *Let f be a nonconstant read-once formula, and let T' be a set of literals. Then T' is a subset of a maxterm of f if and only if every literal in T' occurs in f and for every pair of distinct literals Y and Z in T', $l(lca(X, Y)) =$ OR.*

Proof of Lemma 1. The proof is by induction on the structure of the formula f. If the formula f consists of a single literal Y then the conditions of the lemma reduce to: S is a minterm of f if and only if S is $\{Y\}$, which is correct.

Suppose f consists of the AND of subformulas $f_1, \ldots, f_k$, where the characterization of the lemma holds for all the subsets of minterms of the formulas f_i. If S' is a subset of a minterm of f, S' is the union of sets $S'_1, \ldots, S'_k$ where S'_i is a subset of a minterm of f_i for each i. Let Y and Z be distinct literals in S'. If they both occur in some S'_i, then by the induction hypothesis, $l(lca(Y, Z)) =$ AND. And if Y and Z occur in distinct S'_i and S'_j, then $lca(Y, Z)$ is the root of f, which is labelled AND. In either case, $l(lca(Y, Z)) =$ AND.

Conversely, suppose S' is a set of literals that all occur in f and for each pair Y and Z of distinct literals in S', $l(lca(Y, Z)) =$ AND. Let S'_i be the intersection of S' and the literals that occur in f_i. Then S'_i is a set of literals that all occur in f_i and is such that for every pair of distinct literals Y and Z in S'_i, $l(lca(Y, Z)) =$ AND. Thus by the induction hypothesis, S'_i is a subset of a minterm of f_i, and S' is a subset of a union of a set of minterms for the formulas f_i, so S' is a subset of a minterm for f.

For the last case, suppose f is the OR of subformulas $f_1, \ldots, f_k$, where the characterization of the lemma holds of the subsets of minterms of the formulas f_i. If S' is a subset of a minterm of f, then S' is a subset of a minterm of f_i for some i, so all the literals in S' occur in f_i and for every pair Y and Z of distinct literals from S', $l(lca(Y,Z)) =$ AND.

Conversely, if S' is a set of literals that occur in f and for every pair Y and Z of distinct literals in S', $l(lca(Y,Z)) =$ AND, then in particular all the literals in S' must occur in a single f_i (for otherwise two of them would have the root of f, labelled OR, as their lowest common ancestor), so by the induction hypothesis, S' is a subset of a minterm of f_i. Hence S' is a subset of a minterm of f. Q.E.D.

The proof of Lemma 2 is dual. These two results have the following easy consequences. Let f be a nonconstant read-once formula. Every literal that occurs in f must be an element of both a minterm and maxterm of f. If Y and Z are distinct literals that occur in f then $\{Y, Z\}$ is a subset of a minterm of f if and only if $l(lca(Y,Z)) =$ AND, and $\{Y, Z\}$ is a subset of a maxterm of f if and only if $lca(Y,Z) =$ OR. Therefore, a minterm and a maxterm of f contain at most one literal in common.

Moreover, it is well known that a minterm and a maxterm of *any* boolean function contain at least one literal in common. The proof is simple. Suppose S and T are respectively the minterm and maxterm of a function, and S and T contain no common literals. Then we can set the literals in S to 1 which forces the value of the function to be 1, and at the same time set the literals in T to 0, which forces the value of the function to be 0, a contradiction. We have now proved the following lemma.

Lemma 3 *A minterm and a maxterm of a read-once formula contain exactly one literal in common.*

Karchmer et al. [16] have given an elegant combinatorial characterization of read-once formulas from which Lemma 3 follows, but the derivation above provides some additional insight.

2.5 The procedures *Findmin* and *Findmax*

Let f be a nonconstant monotone read-once function over V_n. Let $V \subseteq V_n$ be such that $f(1_V) = 1$. Then V contains a minterm of f, and there is a simple procedure, *Findmin*, to find a minterm of f contained in V using $O(n)$ membership queries to f and time $O(n)$. The method is a greedy search, removing as many variables from V as possible while preserving the condition that $f(1_V) = 1$.

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

Findmin

1. Set $V_0 = V$.
2. For j=1 to k do:

(a) Set $U_j = V_{j-1} - \{X_{i_j}\}$.

(b) If $f(1_{U_j}) = 1$ then set $V_j = U_j$ else set $V_j = V_{j-1}$.

3. Output V_k.

We claim that V_k is a minterm of f. Note that for each $j = 1, \ldots, k$, V_j is a subset of V_{j-1}, and $X_{i_j} \in V_k$ if and only if $f(U_j) = 0$. Note that $f(1_{V_0}) = 1$ and this condition is preserved at every step, so $f(1_{V_k}) = 1$.

Suppose for some proper subset $S \subset V_k$, $f(1_S) = 1$. Let X_{i_j} be an element of V_k not in S. Then S is a subset of V_{j-1}, and therefore a subset of $U_j = V_{j-1} - \{X_{i_j}\}$. Thus, $1_S \leq 1_{U_j}$, and since f is monotone, $f(1_{U_j}) = 1$. But this means that X_{i_j} is not in V_k, a contradiction. Thus V_k is a minterm of f.

We describe the implementation of *Findmin* to support our claim that its running time is $O(n)$. The input is a list of indices i of elements X_i of a set $V \subseteq V_n$ such that $f(1_V) = 1$. Initially, *Findmin* constructs a representation of the vector 1_V in n consecutive registers in memory, which takes time $O(n)$.

Then, for each index i of an element $X_i \in V$, *Findmin* changes the i^{th} register in the representation to 0, and does a membership query with the resulting representation of a vector. If the reply is 0, it restores the value of the i^{th} register to 1, and if the reply is 1, it does nothing. There are at most n iterations of this process – each one takes one membership query and constant time.

The final vector is converted back into the list of indices where it is equal to 1, which takes time $O(n)$, and this list is returned. Thus, using time $O(n)$ and at most n membership queries, *Findmin* finds a minterm of f contained in V.

A dual procedure, *Findmax*, takes a set V of variables such that $f(0_V) = 0$ and returns a subset T of V such that T is a maxterm of f, using $O(n)$ membership queries to f and time $O(n)$.

3 The minterm graph of a read-once formula

Let f be a read-once formula. Let L' denote the set of literals that occur in f. The *minterm graph* of f is the undirected graph whose vertex set is L' and whose edge set consists of all (Y, Z) such that Y and Z are distinct literals in L' and $\{Y, Z\}$ is a subset of some minterm of f. Equivalently, by Lemma 1, for all Y and Z in L', (Y, Z) is an edge if and only if $l(lca(Y, Z)) =$ AND.

The *maxterm graph* is defined dually as the graph with vertex set L' and edge set consisting of all those (Y, Z) such that Y and Z are distinct vertices in L' and $\{Y, Z\}$ is a subset of some maxterm of f. Thus, by Lemma 2, for all Y and Z in L', (Y, Z) is an edge if and only if $l(lca(Y, Z)) =$ OR. Clearly the maxterm graph of f is the complement of the minterm graph.

Karchmer et al. [16] have shown that if f is nonconstant, then a read-once formula equivalent to f may be recursively constructed from the minterm graph of f as follows. If the graph of f consists of a single literal Y, then f must be equal to the literal Y.

Otherwise, one of the following two cases occurs.

1. The formula f is the OR of the subformulas $f_1, \ldots, f_k$. It follows from Lemmas 1 and 2 that in this case the minterm graph of f is disconnected (into k connected components), and the maxterm graph of f is connected. Moreover, f is equivalent to the OR of the formulas obtained by recursively processing the connected components of the minterm graph of f.

2. The formula f is the AND of the subformulas $f_1, \ldots, f_k$. It follows from Lemmas 1 and 2 that in this case the maxterm graph of f is disconnected (into k connected components), and the minterm graph of f is connected. Moreover, f is equivalent to the AND of the formulas obtained by recursively processing the connected components of the minterm graph of f.

It follows from the above construction that from the minterm graph for f a read-once formula equivalent to f can be constructed in time $O(n^3)$ by repeatedly applying depth-first search to find connected components.

The above construction builds the read-once formula top-down. There is a more complicated, recursive, bottom-up, construction that runs in time $O(n^2)$ [14]. The construction is based on the fact that two literals X and Y are children of the same node in f if and only if for all other literals Z appearing in f, $l(lca(X,Z)) = l(lca(Y,Z))$. The construction uses a trie to store the rows of the adjacency matrix for the minterm graph, and an updating scheme to restore the trie after successive deletions of equal rows. We omit the details here.

4 Using the relevant possibility oracle

As a simple application of the results of the preceding section, we obtain the following theorem.

Theorem 4 *There is a learning algorithm that exactly identifies any read-once formula over V_n in time $O(n^2)$ using $O(n^2)$ relevant possibility queries.*

Proof. The relevant possibility oracle may be used to construct the minterm graph of f. For each literal $Y \in L_n$, one query to the relevant possibility oracle with the set $\{Y\}$ determines whether the literal Y occurs in f.

Then let L' denote the set of literals that occur in f. For each pair Y and Z of distinct literals in L', query the relevant possibility oracle with the set $\{Y, Z\}$. The reply will be "yes" if and only if (Y, Z) is an edge in the minterm graph. Thus, with $O(n^2)$ relevant possibility queries we may construct the minterm graph of f. By the results of the preceding section, a formula equivalent to f may then be obtained in time $O(n^2)$. Q.E.D.

This algorithm improves upon the algorithm of Valiant [26] because it uses only one of the three types of queries used in his algorithm (without increasing the computational complexity or total number of queries). However, it is hard to imagine situations in which relevant possibility queries are directly answered by the environment. As we argued in the introduction, algorithms that use membership and equivalence queries are potentially more applicable. The subsequent sections concern algorithms that use just these two types of queries.

5 Membership queries and monotone read-once formulas

The main result of this section is the following.

Theorem 5 *There is a learning algorithm that exactly identifies any monotone read-once formula over V_n in time $O(n^3)$ using $O(n^2)$ membership queries.*

Thus in the case of monotone read-once formulas, membership queries alone suffice for efficient exact identification. The algorithm has two parts. In the first part, the algorithm generates minterms and maxterms and uses them to determine the set V of variables on which f depends. In the second part, the maxterms generated in the first part are used to determine the label of the lowest common ancestor of each pair of variables in V.

5.1 Finding the variables that occur in f

Assume f is nonconstant. The method of finding all the variables appearing in f is to start with V set to the variables in a single minterm of f and then to perform a closure operation. The closure operation guarantees that every variable in V is contained in at least one minterm and one maxterm composed of variables of V. The following lemma shows that this suffices.

Lemma 6 *Let f be a nonconstant read-once formula. Suppose L is a nonempty set of literals such that for every $Y \in L$, Y is contained in a minterm of f that is a subset of L and in a maxterm of f that is a subset of L. Then L is the set of literals that occur in f.*

Proof. The method is induction on the structure of read-once formulas. In the base case, f consists of a single literal Y. Since $\{Y\}$ is the only minterm and the only maxterm of f, the nonempty set L must be $\{Y\}$.

Suppose now that f is the AND of the subformulas $f_1, \ldots, f_k$ and the assertion of the theorem holds for each f_i. Let L be a nonempty set of literals such that for every $Y \in L$, Y is contained in a minterm of f that is a subset of L and in a maxterm of f that is a subset of L. Let L_i be the intersection of L with the literals occurring in f_i.

Since L is nonempty, it contains at least one literal Y, and Y must be contained in a minterm S of f that is a subset of L. Since the root of f is labelled AND, S is the union of minterms for the formulas f_i, so each L_i is nonempty.

Now consider any literal $Z \in L_i$. Since $L_i \subseteq L$, Z is contained in a minterm S' of f and a maxterm T' of f such that $S' \subseteq L$ and $T' \subseteq L$.

Since the root of f is labelled AND, S' consists of a union of minterms for the formulas f_i, and in particular, Z is contained in a minterm for f_i that is a subset of L_i. Also, T' consists of a maxterm for f_j for exactly one value of j. Since T' contains Z and Z occurs in f_i, T' must be a maxterm for f_i. Hence, Z is also contained in a maxterm for f_i that is a subset of L_i.

Thus, for each i, L_i is nonempty and for each literal $Z \in L_i$, Z is contained in a minterm of f_i that is a subset of L_i and in a maxterm of f_i that is a subset of L_i. Thus, by the induction hypothesis, L_i consists of the literals occurring in f_i. Since this is true for each i, L is the set of literals occurring in f.

For the case in which f is the OR of subformulas $f_1, \ldots, f_k$, we argue dually. Q.E.D.

It is simple to find a single minterm of f. f is nonconstant and monotone, so V_n contains a minterm of f. Therefore, executing Findmin(V_n) will produce a minterm S of f. Given S, we want to implement the closure operation defined above. The implementation of the closure operation depends on the following fact.

Lemma 7 *Let f be a monotone read-once formula over V_n. If S is a minterm of f containing the variable X_i, then $(V_n - S) \cup \{X_i\}$ contains a maxterm of f, and any such maxterm contains X_i. Dually, if T is a maxterm of f containing X_i, then $(V_n - T) \cup \{X_i\}$ contains a minterm of f, and any such minterm contains X_i.*

Proof. If S is a minterm of f containing X_i then the vector that is 1 on $S - \{X_i\}$ and 0 elsewhere assigns 0 to f. Thus $(V_n - S) \cup \{X_i\}$ contains a maxterm for f.

By Lemma 3, every maxterm of f contains exactly one literal in common with S. $(V_n - S) \cup \{X_i\}$ contains only one element of S, namely X_i. Therefore any minterm of f contained in $(V_n - S) \cup \{X_i\}$ must contain X_i.

The argument is dual for the case of a maxterm containing X_i. Q.E.D.

The following algorithm implements the closure operation described in Lemma 6. Given a membership oracle for a nonconstant monotone read-once formula f over V_n, it finds the set V of variables that occur in f. In the process, for each variable X_i in V, it generates a minterm $m[i]$ and a maxterm $M[i]$ containing X_i.

Findvars

1. Initialize $m[i] = M[i] = \emptyset$ for $i = 1, \ldots, n$.
2. Set $V = S = \mathit{Findmin}(V_n)$. For each $X_i \in V$, set $m[i] = S$.
3. While there exists $X_i \in V$ such that exactly one of $M[i]$ and $m[i]$ is $\emptyset$, do:
   (a) If $M[i] = \emptyset$ then set $T = \mathit{Findmax}((V_n - m[i]) \cup \{X_i\})$, set $V = V \cup T$, and for each $X_j \in T$, set $M[j] = T$.

(b) If $m[i] = \emptyset$ then set $S = Findmin((V_n - M[i]) \cup \{X_i\})$, set $V = V \cup S$, and for each $X_j \in S$, set $m[j] = S$.

The correctness of the Findvars algorithm follows from Lemmas 6 and 7. Each call to Findmin and Findmax makes $O(n)$ membership queries. The body of step (3) of Findvars is executed at most $2n$ times, so the total number of membership queries performed by the algorithm is $O(n^2)$. The body of step (3) can be implemented to run in time $O(n)$, so the entire algorithm can be implemented to run in time $O(n^2)$.

5.2 Testing the *lca* of pairs of variables in f

The procedure of the preceding section finds the set V of variables that occur in f, and, for each $X_i \in V$, a minterm and a maxterm of f containing X_i. To construct the minterm graph of f, we need to determine the label of the lowest common ancestor of X_i and X_j for each pair of distinct variables X_i and X_j in V. If we have a maxterm that contains both X_i and X_j, then by Lemma 2 the lowest common ancestor of X_i and X_j is an OR. Otherwise, we use the following fact.

Lemma 8 *Let f be a nonconstant monotone read-once formula over V_n. Suppose T_1 is a maxterm of f containing X_i but not X_j, and T_2 is a maxterm of f containing X_j but not X_i. Let $R = (V_n - (T_1 \cup T_2)) \cup \{X_i, X_j\}$. Then $lca(X_i, X_j)$ is AND if and only if $f(1_R) = 1$.*

Proof. Suppose $f(1_R) = 1$. Then there is a minterm $S \subseteq R$. By Lemma 3 every minterm of f has exactly one literal in common with T_1, and exactly one literal in common with T_2. The set R contains only one element of T_1, namely X_i, and only one element of T_2, namely X_j. Hence $X_i \in S$ and similarly $X_j \in S$, so by Lemma 1 $lca(X_i, X_j)$ is AND.

Conversely, suppose $lca(X_i, X_j)$ is AND. Then there is a minterm S of f such that $X_i \in S$ and $X_j \in S$. Since $X_i \in T_1$, and a minterm and a maxterm of f contain exactly one literal in common, S is disjoint from $T_1 - \{X_i\}$. Similarly, S is disjoint from $T_2 - \{X_j\}$. Hence, S is a subset of R, and, since S is a minterm of f, this implies $f(1_R) = 1$. Q.E.D.

Thus, given the set V of variables occurring in f and for each $X_i \in V$, a maxterm T_i of f containing X_i, we may compute the minterm graph of f by making at most one membership query for every distinct pair of variables X_i and X_j in V. If T_i contains X_j, or T_j contains X_i, then by Lemma 2 $lca(X_i, X_j)$ is OR. Otherwise, simply query the vector that is 1 on

$$(V_n - (T_i \cup T_j)) \cup \{X_i, X_j\}$$

and 0 elsewhere; $lca(X_i, X_j)$ is AND if the reply is 1 and OR if the reply is 0.

Hence, with $O(n^2)$ additional queries, we can determine the minterm graph of f, and construct f from its minterm graph in time $O(n^2)$ as in Section 3. However, the cost of setting up each of the $O(n^2)$ membership queries in this phase is $O(n)$, so the total running time of this phase is $O(n^3)$.

5.3 The algorithm *MM*

The complete algorithm to learn monotone read-once formulas using membership queries is now simple to describe. This algorithm, and the proof of its correctness, constitute the proof of Theorem 5.

MM

1. Use a membership query to test whether $f(1_{V_n}) = 0$. If so, output $\perp$.
2. Use a membership query to test whether $f(0_{V_n}) = 1$. If so, output $\top$.
3. Use the procedure *Findvars* to find the set V of variables occurring in f, and for each $X_i \in V$, a maxterm $M[i]$ of f.
4. For each pair of distinct variables X_i and X_j in V, if $M[i]$ does not contain $M[j]$, and $M[j]$ does not contain X_i, then use a membership query to determine whether $f(1_{R_{i,j}}) = 1$, where $R_{i,j} = (V_n - (M[i] \cup M[j])) \cup \{X_i, X_j\}$, and include an edge (X_i, X_j) in G if so.
5. Construct a formula g from the graph G using the procedure of Section 3 and output g.

Let f be a monotone read-once function over the variables V_n. When *MM* is called with a membership oracle for f, the first two steps determine whether f is the constant function $\perp$ or $\top$. Otherwise, f is a nonconstant monotone read-once function, and step (3) finds the variables V that f depends on, and for each $X_i \in V$, a maxterm $M[i]$ of f containing X_i.

Then step (4) constructs the minterm graph G of f and step (5) constructs and outputs a formula g for the function f. By the previous analyses, the total number of membership queries is $O(n^2)$ and the total time is $O(n^3)$. This concludes the proof of Theorem 5.

6 Using membership and equivalence queries

The main result of this section is the following.

Theorem 9 *There is a learning algorithm that exactly identifies any read-once formula over V_n in time $O(n^4)$ using $O(n^3)$ membership queries and $O(n)$ equivalence queries.*

A simple adversary argument shows that no polynomial time algorithm can exactly identify all the read-once formulas over V_n using only membership queries [2]. In section 7 we show that no polynomial time algorithm can exactly identify all the read-once formulas over V_n using just equivalence queries, so both equivalence queries and membership queries are essential to this theorem.

6.1 Determining signs

Suppose f is an arbitrary read-once formula over V_n and we happen to know the set of variables that occur in f and their signs. Then we can use the procedure *MM* as a subroutine to achieve exact identification of f. This is because the value of f is a monotone function of the values assigned to its leaves, and, knowing the signs of the literals, we can calculate the values assigned to the leaves from the values assigned to the variables V_n.

More precisely, for any $y \in B_n$ define

$$(f \oplus y)(x) = f(x \oplus y)$$

for all $x \in B_n$, where $\oplus$ denotes exclusive or. Then it is not difficult to prove that $(f \oplus y)$ is a read-once function, and a formula for $(f \oplus y)$ can be obtained from f by substituting $\neg X_i$ for X_i for every i such that $y[i] = 1$.

The boolean vector $x \in B_n$ is defined to be a *correct sign vector for* f if and only if for every X_i that occurs in f, X_i occurs negatively if and only if $x[i] = 1$. The following lemma is not difficult to prove.

Lemma 10 *If f is an arbitrary read-once formula over V_n and y is a correct sign vector for f then $(f \oplus y)$ is a monotone read-once function.*

Thus, we may formalize a modification of *MM* that learns an arbitrary read-once formula f over V_n provided that a correct sign vector for f is given as input.

MMSigns

1. With boolean vector y as input, call *MM*. Each time *MM* makes a membership query, say with vector x, reply with the value of $f(x \oplus y)$, obtained by making a membership query to f.
2. If *MM* returns the monotone formula g, then return the formula g' obtained by replacing X_i with $\neg X_i$ if and only if $y[i] = 1$.

Lemma 11 *If MMSigns is called with a membership oracle for an arbitrary read-once formula f over V_n and a correct sign vector y for f, it exactly identifies f in time $O(n^3)$ using at most $O(n^2)$ membership queries.*

Proof. By Lemma 10, $(f \oplus y)$ is a monotone read-once function. By the proof of Theorem 5, the algorithm *MM* exactly identifies any monotone read-once formula over V_n in time $O(n^3)$ using $O(n^2)$ membership queries. $f(x \oplus y) = (f \oplus y)(x)$, so step (1) of *MMSigns* actually runs the algorithm *MM* with the membership oracle for $(f \oplus y)$. The monotone read-once formula g

returned by *MM* expresses the function $(f \oplus y)$. So clearly the formula g' returned in step (2) of *MMSigns* is equivalent to f. Q.E.D.

It follows from Lemma 11 that if we knew the signs of the variables occurring in f, we could identify f using membership queries. However, initially we have no knowledge of the variables occurring in f or of their signs. The following lemma indicates how signs of variables in f may be discovered.

Lemma 12 *Let f be an unknown read-once formula over V_n. Suppose y_0 and y_1 are boolean vectors such that $f(y_0) = 0$ and $f(y_1) = 1$. Then with at most n membership queries to f we can determine the sign of a variable X_i occurring in f such that $y_0[i] \neq y_1[i]$.*

Proof. Let k be the number of bit positions in which y_1 differs from y_0. We use a *bit flipping* technique. We transform y_0 into y_1 bit by bit, that is, we find a sequence of vectors $z_0, z_1, \ldots, z_k$ such that $z_0 = y_0$, $z_k = y_1$, and z_{j+1} differs from z_j in exactly one bit position for $j = 0, \ldots, k-1$. By making at most $k-1$ membership queries with $z_1, z_2, \ldots$, we find a pair of vectors z_j and z_{j+1} such that $f(z_j) = 0$ and $f(z_{j+1}) = 1$. Let i be the bit position where z_j and z_{j+1} differ. Note that y_0 and y_1 must differ in position i as well.

Now if $z_j[i] = 0$ then X_i must occur positively in f, and if $z_j[i] = 1$ then X_i must occur negatively in f. To see this, note that X_i must occur in f, and the value of f is a monotone function of the values assigned to its leaves.

If the literal $\neg X_i$ labels a leaf, then changing the value of X_i from 0 to 1 changes the value assigned to the leaf from 1 to 0, so the value of f could not change from 0 to 1, since this would not be a monotone change. Hence, if $z_j[i] = 0$, then X_i must occur positively in f. Similarly, if $z_j[i] = 1$ then X_i must occur negatively in f. Q.E.D.

We note that the bound in Lemma 12 can be reduced to $\log n$ membership queries by using a technique analogous to binary search.

By Lemma 12, if we can succeed in eliciting the "right" pairs of vectors y_0 and y_1, we can efficiently learn the signs of the variables occurring in f. This is where equivalence queries prove useful; the following example is intended to provide some intuition.

Consider the initial state, in which nothing is known about f. Choose an arbitrary boolean vector x and make a membership query to discover the value of $f(x)$. If $f(x) = 0$, then make an equivalence query with $\perp$. The reply will be either "yes" or a vector x_1 such that $f(x_1) = 1$. If $f(x) = 1$, then make an equivalence query with $\top$. The reply will be either "yes" or a vector x_0 such that $f(x_0) = 0$.

If f is nonconstant then with one equivalence query we have elicited a pair of vectors y_0 and y_1 such that $f(y_0) = 0$ and $f(y_1) = 1$. By the above lemma, we can discover the sign of at least one variable in f. The algorithm *MEQ* continues this process. It first uses the procedure *MMSigns* as a subroutine to identify a projection of the unknown function f for which the signs of the variables are already known. It then makes an equivalence query. If the answer to the equivalence query is "yes", then the projection is equivalent to f. If the answer to the equivalence query is "no" then a counterexample is returned, and the algorithm uses the bit flipping technique on the counterexample to elicit a previously unknown sign of a variable in f.

6.2 Projections of f

Let f be a read-once formula over V_n. A *partial assignment* is a vector a from $\{0, 1, *\}^n$. The *defined set* of a is the set of X_i such that $a[i] \neq *$. Each partial assignment a defines a map from B_n to B_n as follows. For all $x \in B_n$, a/x is the vector y such that if $a[i] \neq *$ then $y[i] = a[i]$, and if $a[i] = *$ then $y[i] = x[i]$. That is, a is used to assign values to the variables X_i such that $a[i] \neq *$, and x is used to assign values to the rest of the variables.

Each partial assignment a induces a *projection* f_a of f defined by

$$f_a(x) = f(a/x)$$

for all $x \in B_n$. It is not difficult to show that every projection of a read-once function is a read-once function. Moreover, every variable that occurs in f_a occurs in f with the same sign. Also note that if a is given, we may simulate membership calls to f_a using membership calls to f.

6.3 The algorithm *MEQ*

We now prove Theorem 9 by describing and analyzing the algorithm *MEQ*.

Proof of Theorem 9. The algorithm *MEQ*, shown below, keeps a set W containing the variables in f whose signs have been learned. The algorithm records the signs of the variables in W in a vector x_W, by setting $x_W[i] = 1$ if and only if the sign of X_i in f is known to be negative.

MEQ

1. Let $W = \emptyset$ and $x_W[i] = 0$ for all $i = 1, \ldots, n$.
2. Do forever:
   (a) Let a be an arbitrary partial assignment whose defined set is $V_n - W$.
   (b) Call the procedure *MMSigns* with input vector x_W, simulating membership queries to the function f_a, and 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 y be the counterexample.
   (d) In this case, $g(y) = f_a(y) \neq f(y)$, that is, $f(a/y) \neq f(y)$, so use the bit flipping technique to find the sign of some $X_i \notin W$. Add X_i to W and set $x_W[i] = 1$ if the sign of X_i is negative.

We now prove the correctness of *MEQ* and analyze its running time.

Let f be an arbitrary read-once formula over V_n, and suppose *MEQ* is called with membership and equivalence oracles for f. We will show that every time execution reaches step (2a), W and x_W have the following property:

1. For each i such that $X_i \in W$, X_i occurs in f negatively if $x_W[i] = 1$ and positively if $x_W[i] = 0$.

Moreover, we will show that every nonterminating execution of the body of step (2) adds a new element to W.

Property (1) is certainly true the first time execution reaches step (2a), since W is the empty set. Assume that property (1) is true before some execution of step (2a). The partial assignment a chosen in step (2a) assigns values to all the variables not in W, so f_a is a read-once function such that every variable f_a depends on is in W, and if X_i occurs negatively in f_a, then $x_W[i] = 1$ and if X_i occurs positively in f_a, then $x_W[i] = 0$. Thus, by Lemma 11, when *MMSigns* is called with input vector x_W and a (simulated) membership oracle for f_a, it exactly identifies f_a in time $O(n^3)$ using $O(n^2)$ membership queries. Hence the formula g is equivalent to f_a in step (2b).

Clearly, if the equivalence query in step (2c) is answered "yes", then *MEQ* terminates correctly. Otherwise, the counterexample y has the property that $g(y) \neq f(y)$. Since g is equivalent to f_a and $f_a(y) = f(a/y)$, this implies that $f_a(y) \neq f(y)$.

Thus in step (2d) we have two vectors, y and a/y, on which f takes two different values, so, by Lemma 12, using at most n membership queries we can determine the sign of one variable X_i in f such that $y[i] \neq a/y[i]$. Thus X_i is in the defined set of a, and therefore X_i is not in W. When X_i is added to W and x_W is updated with the sign of X_i, the values of W and x_W again satisfy property (1), and a new element has been added to W.

Thus, each nonterminating execution of the body of step (2) determines one previously unknown sign of a variable in f. By Lemma 11, when W contains all the variables occurring in f, the call to *MMSigns* must return a formula g equivalent to f. Thus, *MEQ* must terminate correctly after at most n nonterminating iterations of step (2). Hence, after at most $n + 1$ equivalence queries, at most $O(n^3)$ membership queries, and time at most $O(n^4)$, *MEQ* must exactly identify f. Q.E.D.

A simple corollary of this result is the existence of a polynomial time algorithm to identify read-once formulas using subset and superset queries. The input to a *subset query* is a read-once formula g and the reply is "yes" if g logically implies f. Otherwise, the reply is a vector x such that $h(x) = 1$ and $f(x) = 0$. A *superset query* is defined dually, to test whether h is logically implied by f. Since one subset query with a conjunction of n literals can be used to answer a membership query, and a pair consisting of a subset query and a superset query can be used to answer an equivalence query, we have the following result, independently proved by Hancock [10].

Corollary 13 *There is an algorithm that exactly identifies all the read-once formulas over V_n in time polynomial in n using subset and superset queries.*

7 A generalization of this transformation

The transformation given in the preceding section of the algorithm *MM* into the algorithm *MEQ* can be usefully generalized. In this section we prove a generalization that implies as a corollary that the class of unate DNF formulas can be exactly identified in polynomial time using membership and equivalence queries.

A boolean function f is *unate* if and only if for no variable X_i does X_i occur positively in some minterm of f and negatively in some other minterm of f (The term *unate* is used in switching theory). Every read-once function is unate, and every monotone boolean function is unate, but there are unate functions that are neither read-once nor monotone.

Let f be a unate boolean function over V_n. The variable X_i *occurs in* f if and only if a literal of X_i is an element of some minterm of f. If X_i occurs in f, then the *sign of* X_i *in* f is the sign of the literal of X_i that occurs in a minterm of f. The vector $y \in B_n$ is a *correct sign vector for* f if and only if for every X_i that occurs in f, $y[i] = 1$ if and only if the sign of X_i in f is negative.

Let M be a class of monotone boolean formulas. M is *closed under projection* if for every formula $f \in M$ on V_n and every partial assignment $a \in \{0, 1, *\}^n$, there is a formula in M equivalent to f_a.

If f is any monotone boolean formula over V_n, let $U(f)$ denote the class of all formulas f' obtained from f by selecting a subset V of V_n and replacing every occurrence of X_i in f by $\neg X_i$. Note that all the elements of $U(f)$ are unate. If M is a class of monotone boolean formulas, let $U(M)$ denote the union of $U(f)$ for all $f \in M$. Observe that if M is the class of monotone read-once formulas, then $U(M)$ is the class of general read-once formulas.

Note that the following theorem strengthens the transformation in the previous section by allowing equivalence queries to be used in the identification algorithm for M.

Theorem 14 *Let M be a class of monotone formulas that is closed under projection, and suppose that there is a polynomial time algorithm that exactly identifies every element of M using membership and equivalence queries. Then there is a polynomial time algorithm that exactly identifies every element of $U(M)$ using membership and equivalence queries.*

Proof. Let A be a polynomial time algorithm that exactly identifies every formula in M using membership and equivalence queries. As before, we can use A to identify an element f of $U(M)$ provided we know the signs of the variables that occur in f.

In particular, suppose f is a formula from $U(M)$ over V_n and z is a correct sign vector for the function represented by f. Then we can identify f using membership and equivalence queries with A as a subroutine as follows.

Begin running A. When A makes a membership query with the element x, return the value $f(x \oplus z)$, using a membership query for f. When A makes an equivalence query with the formula g, make an equivalence query with the formula g' obtained from g by replacing every occurrence of X_i in g by $\neg X_i$ if $z[i] = 1$. If the reply is "yes", then return the reply "yes" to A. Otherwise, the reply is a counterexample y such that $g'(y) \neq f(y)$. In this case,

$$g(y \oplus z) = g'(y) \neq f(y) = (f \oplus z)(y \oplus z),$$

so return the counterexample $y \oplus z$ to A. When A terminates with the formula g, terminate with the formula g' for $(g \oplus z)$.

Thus, by a proof similar to that in the previous section, there is a polynomial time algorithm *ASigns* that identifies every element f of $U(M)$ using membership and equivalence queries, provided it is given as input a correct sign vector for f. As in the preceding section, we use this algorithm to attempt to identify projections of f for which we know the signs; the major difference is that we must now handle equivalence queries made by *ASigns*.

The algorithm AU to identify an element $f \in U(M)$ over V_n works as follows. Let W be the set of variables of known sign in f, and let x_W be the vector that is 1 if and only if the sign of X_i is known to be negative. Initially W is the empty set and x_W is the vector of all zeroes.

AU chooses an arbitrary partial assignment a whose defined set is $V_n - W$, and calls the procedure *ASigns* with the input vector x_W. For each membership query of *ASigns*, say with element y, AU returns the value of $f_a(y) = f(a/y)$, using a membership query to f.

If *ASigns* makes an equivalence query, say with formula g, we call the equivalence oracle for f with formula g. If the reply is "yes", then we output g and halt, since we have found a formula in $U(M)$ equivalent to f. Otherwise, the reply is a counterexample y such that $f(y) \neq g(y)$.

Using the membership oracle for f, we can compute $f_a(y)$. We know g, so we can compute $g(y)$. If $f_a(y) \neq g(y)$, then we return y as the counterexample to *ASigns*. Otherwise, $g(y) = f_a(y) \neq f(y)$, so by Lemma 12 we can use at most n membership queries to f to determine the sign of a variable X_i in the defined set of a, that is, not in W. In this case, we add X_i to W and update its sign in x_W, give up the current simulation of *ASigns* and iterate the loop with the new values of W and x_W.

If we succeed in answering all the queries of *ASigns*, then it will return with a formula g that is equivalent to f_a. We make an equivalence query with g. If the reply is "yes", then we output g and terminate. Otherwise the reply is a counterexample y such that

$$f(y) \neq g(y) = f_a(y) = f(a/y),$$

so, as before, we determine the sign of a variable X_i not in W, add X_i to W, update the sign of X_i in x_W, and iterate.

The algorithm AU correctly identifies every element of $U(M)$ over V_n using at most n calls to the original algorithm A, at most one membership query for each equivalence query made by A, and an additional $O(n)$ equivalence queries and $O(n^2)$ membership queries. Q.E.D.

Let M denote the class of monotone DNF formulas. Then M can be exactly identified in polynomial time using membership and equivalence queries [2], and M is closed under projection. Moreover, $U(M)$ is the class of unate DNF formulas. Thus, as an easy corollary of the theorem above, we have the following.

Corollary 15 *The class of unate DNF formulas is exactly identifiable in polynomial time using membership and equivalence queries.*

8 The insufficiency of equivalence queries

In this section we show that no polynomial time algorithm can exactly identify all the read-once formulas using only equivalence queries, even if the target formula is known to be monotone and in disjunctive normal form. The proof is similar to the proof in [4] that there is no polynomial time algorithm to identify all DNF formulas using only equivalence queries. That proof made use of a certain combinatorial property of DNF formulas: every DNF formula is satisfied by an assignment with "few" ones or falsified by an assignment with "few" zeroes. An analogous property for read-once formulas is now proved.

8.1 A bound on the sizes of minterms and maxterms

Lemma 16 *Let f be any nonconstant read-once formula. Let $m_S(f)$ denote the minimum cardinality of any minterm of f, and $m_T(f)$ denote the minimum cardinality of any maxterm of f. Then $m_S(f)m_T(f) \leq size(f)$.*

Proof. This is proved by induction on the size of f. The base case is a formula f consisting of a single literal. In this case, $m_S = m_T = size(f) = 1$, and the result holds.

Assume that the result holds for any nonconstant read-once formula of size at most $t-1$, and let f be a nonconstant read-once formula of size $t \geq 2$. Then f is the conjunction or disjunction of $k \geq 2$ subformulas $f_1, \ldots, f_k$. Clearly, $size(f_i) \leq t-1$ for each $i = 1, \ldots, k$.

Suppose f is the conjunction of $f_1, \ldots, f_k$. Then a minterm for f consists of the union of a set of minterms for the formulas f_i, so

$$m_S(f) = m_S(f_1) + \ldots + m_S(f_k).$$

A maxterm for f consists of a maxterm for any one of the formulas f_i, so

$$m_T(f) = \min\{m_T(f_1), \ldots, m_T(f_k)\}.$$

Since for each $i = 1, \ldots, k$,

$$\min\{m_T(f_1), \ldots, m_T(f_k)\} \leq m_T(f_i),$$

we have

$$m_S(f)m_T(f) \leq m_S(f_1)m_T(f_1) + \ldots + m_S(f_k)m_T(f_k).$$

By the inductive assumption, $m_S(f_i)m_T(f_i) \leq size(f_i)$ for each $i = 1, \ldots, k$, so

$$m_S(f)m_T(f) \leq size(f_1) + \ldots + size(f_k) = size(f).$$

The case of f being the disjunction of $f_1, \ldots, f_k$ is dual. Q.E.D.

Corollary 17 follows almost immediately from Lemma 16. The corollary does not hold in the case $n = 1$ (the counterexample is $f = \neg x$).

Corollary 17 *Let f be a nonconstant read-once formula of size $n > 1$ and let $m = \lfloor \sqrt{n} \rfloor$. Then f is satisfied by an assignment with exactly m ones or is falsified by an assignment with exactly m zeroes.*

Proof. Since $n > 1$, $n - m \geq m$. By the preceding lemma, f has a minterm or maxterm of cardinality at most m.

Suppose f has minterm S such that $|S| \leq m$. Let P be the set of variables that occur positively in S. Let Q be a set of $m - |P|$ variables from $V_n - S$. Let x_S be the vector that assigns 1 to every variable in $P \cup Q$ and 0 to every other variable. Then clearly x_S has exactly m ones and assigns 1 to every literal in S, so $f(x_S) = 1$.

The case of f having a maxterm of cardinality at most m is dual. Q.E.D.

8.2 The target class of read-once formulas

Let $m \geq 1$ and let $n = m^2$. Let H_n be the class of all monotone read-once DNF formulas with exactly m literals in each monomial. There is a one-to-one correspondence of elements of H_n with permutations of the n variables. That is, each element of H_n can be imagined as the result of taking a permutation of the variables and grouping them, in order, into m monomials of size m each. Thus, $|H_n| = n!$.

Each element of H_n is logically equivalent to those elements that can be obtained by permuting the monomials and by permuting the variables within a monomial. Each element of H_n is logically equivalent to $(m!)^{m+1}$ elements of H_n. Thus, the elements of H_n represent $n!/(m!)^{m+1}$ logically distinct functions. Clearly, for $m \geq 2$, $n!/(m!)^{m+1} \geq 2$.

Suppose $x \in B_n$ is a boolean vector with exactly m ones. Let P_x be the set of variables X_i such that $x[i] = 1$. How many elements $f \in H_n$ have $f(x) = 1$? In order to have $f(x) = 1$, one of the monomials of f must contain exactly the variables in P_x. There are m choices for which monomial this is, and $m!$ permutations of P_x to constitute it. For the remaining $n - m$ variables, there are $(n - m)!$ permutations giving the order in which they are placed in the remaining $n - m$ positions. Thus, there are

$$m \cdot m! \cdot (n - m)!$$

elements $f \in H_n$ such that $f(x) = 1$.

Suppose that $x \in B_n$ is a boolean vector that contains exactly m zeroes. Let N_x be the set of variables X_i such that $x[i] = 0$. How many elements $f \in H_n$ are such that $f(x) = 0$? In order to have $f(x) = 0$, each monomial of f must contain at least one element of N_x. However, since there are m elements of N_x and m monomials that contain disjoint sets of variables, this means that each monomial of f must contain exactly one element of N_x. There are m^m choices of one place in each monomial for the elements of N_x, and $m!$ permutations of N_x by which they might be filled. For the remaining $n - m$ elements, there are $(n - m)!$ orders in which to put them into the remaining $n - m$ places. Thus, there are

$$m^m \cdot m! \cdot (n - m)!$$

elements $f \in H_n$ such that $f(x) = 0$.

The following bound will be used in the proof.

Lemma 18 *For any constant $C > 1$ and for all sufficiently large integers m, if $n = m^2$ then*

$$(m^m \cdot m! \cdot (n - m)!)/n! \leq C\sqrt{2\pi(m - 1)}e^{-(m-1)}.$$

Proof. Let $C > 1$ be given. Choose $\epsilon > 0$ sufficiently small that

$$(1 + \epsilon)^2/(1 - \epsilon) \leq C.$$

By Stirling's approximation to the factorial, we may choose $m > 1$ sufficiently large that for all $k \geq m$,

$$(1 - \epsilon)\sqrt{2\pi k}(k^k/e^k) \leq k! \leq (1 + \epsilon)\sqrt{2\pi k}(k^k/e^k).$$

Let $n = m^2$. Then, since $n \geq (n - m) \geq m$, we have after some simplification,

$$(m^m \cdot m! \cdot (n - m)!)/n! \leq C\sqrt{2\pi(m - 1)}(1 - m/n)^{n-m}.$$

Using the fact that $(1 - x) \leq e^{-x}$, we have

$$(m^m \cdot m! \cdot (n - m)!)/n! \leq C\sqrt{2\pi(m - 1)}e^{-(m-1)}.$$

Q.E.D.

8.3 Equivalence queries do not suffice

Theorem 19 *There is no polynomial time algorithm that exactly identifies all read-once formulas using equivalence queries. In fact, there is no polynomial time algorithm that exactly identifies the class of monotone read-once formulas in disjunctive normal form, even if the equivalence queries are allowed to consist of arbitrary read-once formulas.*

Proof. Suppose to the contrary that A is a polynomial time algorithm that exactly identifies all read-once formulas using equivalence queries. Let $p(n)$ be a polynomial bounding the running time of A. Let $C > 1$, and let $B(m)$ be the function defined by

$$B(m) = C\sqrt{2\pi(m-1)}e^{-(m-1)}.$$

Choose $m > 1$ sufficiently large that for $n = m^2$ we have

$$(m^m \cdot m! \cdot (n-m)!)/n! \leq B(m),$$

(by Lemma 18) and also

$$n!(1 - B(m)p(n)) > (m!)^{m+1}.$$

This latter is possible by our earlier observation that $n!/(m!)^{m+1} \geq 2$, and the fact that for $n = m^2$, $B(m)p(n) \to 0$ as $m \to \infty$.

Let $n = m^2$. Now consider the following adversary strategy. Run algorithm A with variable set $\{X_1, \ldots, X_n\}$ until it makes an equivalence query or has run for $p(n)$ steps, whichever comes first. If it makes an equivalence query with the read-once formula g, then we answer as follows.

If g is $\top$, the reply is "no" and the counterexample is the vector of all 0's. If g is $\bot$, the reply is "no" and the counterexample is the vector of all 1's. Otherwise, g is a nonconstant read-once formula of size at most n. By Corollary 17, since $n > 1$, there is a vector x such that x contains exactly m ones and $g(x) = 1$, or x contains exactly m zeroes and $g(x) = 0$. In either case, the reply is "no" and the counterexample is x.

Consider now the target set H_n. We argue that after $p(n)$ steps of A, at least two logically inequivalent members of H_n are consistent with all the replies given to equivalence queries. To see this, note that each element of H_n is a nonconstant read-once formula, so the counterexamples given in response to $\top$ or $\bot$ do not eliminate any elements of H_n.

In case the counterexample is a vector x with m 1's such that $g(x) = 1$, then we have argued above that there are at most

$$m \cdot m! \cdot (n-m)!$$

elements $f \in H_n$ such that $f(x) = 1$. By our choice of m, since $m \leq m^m$, this means that no more than $B(m)n!$ elements are eliminated from H_n by this counterexample.

In the case that the counterexample is a vector x with m 0's such that $g(x) = 0$, we have argued above that there are at most

$$m^m \cdot m! \cdot (n-m)!$$

elements $f \in H_n$ such that $f(x) = 0$. By our choice of m, this means that at most $B(m)n!$ elements are eliminated from H_n by this counterexample.

Thus, each counterexample eliminates at most $B(m)n!$ elements from H_n. Since H_n initially contains $n!$ elements, this means that when A has run for no more than $p(n)$ steps, there are at least

$$n! - p(n)B(m)n! = n!(1 - p(n)B(m)) > (m!)^{m+1}$$

elements remaining in H_n that are consistent with all the counterexamples returned to this point.

Since each element of H_n is logically equivalent to $(m!)^{m+1}$ elements of H_n, this means that at least two logically inequivalent elements of H_n are consistent with all the replies to this point. Hence, A must either run for more than $p(n)$ steps, or it must fail to identify correctly at least one of the elements of H_n. This contradiction shows that no such A can exist. Note that the set H_n contains only read-once formulas that are monotone and in disjunctive normal form, and the equivalence queries consist of arbitrary read-once formulas. Q.E.D.

9 Summary and remarks

Table 1 summarizes what is known of the computational difficulty of learning monotone and arbitrary read-once formulas according to six types of learning protocols. The entries are discussed in order below.

The results of Pitt and Valiant [22] show that probably approximately correct identification of monotone read-once formulas is not possible in polynomial time if RP $\neq$ NP. This depends on the fact that the hypotheses output by the algorithm are constrained to be read-once formulas.

The results of Kearns, Li, Pitt and Valiant [19] show that there is a polynomial time reduction of the problem of predicting arbitrary boolean formulas to the problem of predicting monotone read-once formulas. The results of Kearns and Valiant [18, 20] give a polynomial time reduction of three apparently hard cryptographic problems to the problem of predicting arbitrary boolean formulas. These results apply also to the problem of probably approximately correctly identifying monotone read-once formulas using arbitrary polynomial time hypotheses.

Theorem 19 shows that there is no polynomial time algorithm that exactly identifies arbitrary read-once formulas using equivalence queries, even if the target class is known to be monotone and in disjunctive normal form.

Theorem 5 shows that monotone read-once formulas can be exactly identified in time $O(n^3)$ using $O(n^2)$ membership queries. The running time bound has been improved to $O(n^2)$ by Raghavan and Schach [24]. A simple adversary argument [2] shows that there is no polynomial time algorithm that exactly identifies arbitrary read-once formulas using only membership queries.

When both membership and equivalence queries are available, the results for the monotone case are clearly no worse than when just membership queries are available. Theorem 9 shows that there is an algorithm that exactly identifies arbitrary read-once formulas in time $O(n^4)$ using $O(n^3)$ membership and $O(n)$ equivalence queries. Raghavan and Schach's improvement of the running time bound for the monotone case carries over via the transformation in Theorem 14 to improve the running time bound for arbitrary read-once formulas to $O(n^3)$.

Theorem 4 shows that there is an algorithm that exactly identifies arbitrary read-once formulas in time $O(n^2)$ using $O(n^2)$ relevant possibility queries.

Using an information theoretic argument based on the number of monotone read-once formulas

Type of Learning Protocol	*Monotone Read-Once Formulas*	*Arbitrary Read-Once Formulas*
Examples Oracle	No (if RP $\neq$ NP)	No (if RP $\neq$ NP)
Prediction	$\equiv$ *Boolean Formulas*	$\equiv$ *Boolean Formulas*
Equivalence Only	No	No
Membership Only	Yes $O(n^2)$ queries $O(n^2)$ time	No
Membership and Equivalence	Yes $O(n^2)$ queries $O(n^2)$ time	Yes $O(n^3)$ queries $O(n^3)$ time
Relevant Possibility	Yes $O(n^2)$ queries $O(n^2)$ time	Yes $O(n^2)$ queries $O(n^2)$ time

Table 1: Existence of Polynomial Time Learning Algorithms

over n variables, it is not difficult to show that any algorithm that exactly identifies all the read-once formulas using just membership queries must make at least $\Omega(n \log n)$ queries. The algorithm *MM* uses $O(n^2)$ queries: it would be interesting to close the gap between these two bounds.

Another open question is whether it is possible to speed up our algorithms by using randomization and parallelism. This question connects in an interesting way to the problem of finding upper and lower bounds on random and parallel algorithms for learning a minterm of a monotone read-once function using membership queries. Lower bounds on random and parallel algorithms for learning a minterm of an arbitrary monotone function using membership queries can be derived from the lower bounds for independence system oracles of Karp, Upfal and Wigderson [17].

Various interesting generalizations of the results in this paper have been obtained. Raghavan and Schach [24] give a polynomial time algorithm using membership and equivalence queries to learn the class of *switch configurations*, which generalizes the class of monotone read-once formulas. Their results imply an $O(n^2)$ time algorithm for learning monotone read-once formulas using $O(n^2)$ membership queries, which improves our time bound by a factor of n. Hancock [11] gives a polynomial time algorithm using membership and equivalence queries to learn the class of μ-formula decision trees, which generalizes the class of read-once formulas. Hancock and Hellerstein [13] give polynomial time algorithms using membership and equivalence queries to learn read-once formulas over extended bases including negation, threshold gates, and gates computing congruence to a modulo c, where c is bounded by a fixed constant.

Another possible direction for generalization is to allow at most k occurrences of each variable in the target formula; this defines the class of $k\mu$-formulas. For $k \geq 3$, Angluin and Kharitonov show that predicting $k\mu$-formulas with membership queries is no easier (modulo a polynomial time transformation) than predicting general boolean formulas with membership queries, and give cryptographic evidence for the difficulty of the latter problem [6]. This leaves open the case of $k = 2$. Hancock gives a polynomial time algorithm for predicting 2μ-DNF using membership queries [12]. Angluin and Kharitonov, and independently, Hancock, show that predicting 3μ-DNF with membership queries is as hard (modulo a polynomial time transformation) as predicting general DNF with membership queries. The complexity of predicting general DNF (or CNF) with membership queries is open, but Angluin and Kharitonov give cryptographic evidence that it is no easier than predicting them without membership queries.

Hancock [10] has shown that read-once formulas and μ-decision trees are identifiable in polynomial time using *constrained instance* queries. (The result for read-once formulas was proved independently by Hellerstein and Karpinski [15].) The input to a constrained instance query is a partial assignment a and a boolean value b, and the reply is "yes" if and only if there exists an assignment x agreeing with a such that $f(x) = b$. Thus, constrained instance queries are very restricted kinds of subset and superset queries, so one corollary of Hancock's results is Corollary 13.

The set of classes identifiable in polynomial time using constrained instance queries appears to be incomparable to the set of classes identifiable in polynomial time using membership and equivalence queries. In particular, the class of monotone DNF formulas is identifiable in polynomial time using membership and equivalence queries, but not with membership queries alone [2]. For classes of monotone formulas, constrained instance queries can be simulated using membership queries

[8], which implies that the class of monotone DNF formulas cannot be identified in polynomial time using constrained instance queries alone. For the other direction, the techniques of Angluin and Kharitonov [6] can be used to give cryptographic evidence that there are classes of formulas identifiable in polynomial time using constrained instance queries that are not identifiable in polynomial time using membership and equivalence queries. Thus, the results of this paper for general read-once formulas appear to be incomparable with those of Hancock.

10 Acknowledgements

It is a pleasure to thank Sally Floyd, Richard Karp, Lenny Pitt, Ron Rivest, Les Valiant, and Manfred Warmuth for helpful discussions of this material.

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

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

[3] D. Angluin. Using queries to identify μ-formulas. Technical report, Yale University, YALE/DCS/RR-694, 1989.

[4] D. Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[5] D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of Horn clauses. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 186–192. IEEE, 1990.

[6] D. Angluin and M. Kharitonov. When won't membership queries help? In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 444–454. ACM Press, 1991.

[7] D. Angluin and L. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. Syst. Sci.*, 18:155–193, 1979.

[8] A. Blum, L. Hellerstein, and N. Littlestone. Learning in the presence of finitely or infinitely many irrelevant attributes. To appear, Proceedings of the Fourth Annual Workshop on Computational Learning Theory, 1991.

[9] D. Yu. Grigoriev, M. Karpinski, and M. F. Singer. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *Siam J. Comput.*, pages 1059–1063, 1990.

[10] T. Hancock. Identifying μ-decision trees and μ-formulas with constrained instance queries. Manuscript, Harvard University, 1989.

[11] T. Hancock. Identifying μ-formula decision trees with queries. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 23–37. Morgan Kaufmann Publishers, 1990.

[12] T. Hancock. Learning 2μDNF formulas and $k\mu$ decision trees. To appear, Proceedings of the Fourth Annual Workshop on Computational Learning Theory, 1991.

[13] T. Hancock and L. Hellerstein. Learning read-once formulas over fields and extended bases. To appear, Proceedings of the Fourth Annual Workshop on Computational Learning Theory, 1991.

[14] L. Hellerstein. *On Characterizing and Learning Some Classes of Read-once Functions*. PhD thesis, University of California, Berkeley, 1989.

[15] 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.

[16] 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.*

[17] R.M. Karp, E. Upfal, and A. Wigderson. Are search and decision problems computationally equivalent? In *Proc. 17th ACM Symposium on Theory of Computing*, pages 285–295. ACM, 1985.

[18] M. Kearns. *The Computational Complexity of Machine Learning*. PhD thesis, Harvard University, 1989. To be published by MIT Press in the ACM Distinguished Dissertation Series.

[19] M. Kearns, M. Li, L. Pitt, and L. Valiant. On the learnability of boolean formulae. In *Proc. 19th ACM Symposium on Theory of Computing*, pages 285–295. ACM Press, 1987.

[20] M. Kearns and L. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *Proc. 21st ACM Symposium on Theory of Computing*, pages 433–444. ACM Press, 1989.

[21] N. Littlestone. Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[22] L. Pitt and L. Valiant. Computational limitations on learning from examples. *J. ACM*, 35:965–984, 1988.

[23] L. Pitt and M. Warmuth. Reductions among prediction problems: On the difficulty of predicting automata. In *Proceedings of the Third Annual Structure in Complexity Theory Conference*, pages 60–69. IEEE Computer Society Press, 1988.

[24] V. Raghavan and S. Schach. Learning switch configurations. In *Proceedings of Third Annual Workshop on Computational Learning Theory*, pages 38–51. Morgan Kaufmann Publishers, 1990.

[25] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, pages 223–242, 1990.

[26] L. G. Valiant. A theory of the learnable. *C. ACM*, 27:1134–1142, 1984.