Yale University
Department of Computer Science

Malicious Membership Queries and Exceptions

Dana Angluin Martinch Krikis

YALEU/DCS/TR-1019
March 1994

This research was supported by the National Science Foundation, grant CCR-9213881.

Malicious Membership Queries and Exceptions

Dana Angluin Mārtiņš Kriķis
Yale University*

Abstract

We consider two issues in polynomial-time exact learning of concepts using membership and equivalence queries: (1) errors in the answers to membership queries and (2) learning finite variants of concepts drawn from a learnable class.

To study (1), we introduce a malicious membership query, in which errors are permitted on a set of strings in the domain, such that the number of strings plus the sum of their lengths is bounded by L. Equivalence queries are answered correctly, and algorithms are allowed time polynomial in the usual parameters and L. We present a new polynomial-time learning algorithm in this model for monotone DNF formulas.

To study (2), we consider classes of concepts that are polynomially closed under finite exceptions and a natural operation to add exception tables to a class of concepts. Applying this operation, we obtain the class of monotone DNF formulas with finite exceptions. We give a new polynomial-time algorithm to learn the class of monotone DNF formulas with finite exceptions using equivalence and membership queries.

Relating (1) and (2), we give a general transformation showing that any class of concepts that is polynomially closed under finite exceptions and is learnable in polynomial time using membership and equivalence queries is also polynomial-time learnable using malicious membership and equivalence queries. Corollaries include the polynomial-time learnability of the following classes using malicious membership and equivalence queries: deterministic finite acceptors, boolean decision trees, and monotone DNF formulas with finite exceptions.

This research was supported by the National Science Foundation, grant CCR-9213881.

*Department of Computer Science, Yale University, P.O. Box 208285, New Haven, CT 06520. E-mail: angluin@cs.yale.edu and krikis@cs.yale.edu

1 Introduction

There is an impressive and growing number of polynomial-time algorithms, many of them quite beautiful and ingenious, to learn various interesting classes of concepts using equivalence and membership queries. To apply such algorithms in practice, researchers will have to overcome a number of problems.

One significant issue is the problem of errors in answers to queries. Previous learning algorithms in the equivalence and membership query model are guaranteed to perform well assuming that queries are answered correctly, but there is often no guarantee that the performance of the algorithm will "degrade gracefully" if that assumption is not exactly satisfied.

A related issue is the assumption that the target concept is drawn from a particular class of concepts, for example, monotone DNF formulas. Even if the target concept is "nearly" a monotone DNF formula, there is typically no guarantee that the learning algorithm will do anything reasonable.

We introduce models addressing these two issues, and demonstrate a useful relationship between them.

The notion of a finite variant of a concept, that is, a concept with a finite set of exceptions, is a unifying theme in this work. Our model of errors in membership queries can be viewed as combining an equivalence oracle for the target concept and a membership oracle for a finite variant of the target concept. Our model of a concept that is "nearly" monotone DNF is one that is represented by a monotone DNF concept with "few" exceptions. We can view the learning problem as one in which the equivalence and membership oracles present the same finite variant of a monotone DNF formula. In both cases, the goal is to identify exactly the concept presented by the equivalence oracle.

1.1 Error Models

There is a considerable body of literature on errors in examples in the PAC model, starting with the first error-tolerant algorithm in the PAC model, given by Valiant [13]. In this case the goal is PAC-identification of the target concept, despite the corruption of the examples by one or another kind of error, for example, random or malicious misclassification errors, random or malicious attribute errors, or malicious

errors (in which both attributes and classification may be arbitrarily changed.)

There has been less work on errors in learning models in which membership queries are available, and the issues are not as well understood. One relevant distinction is whether the errors in answers to membership queries are persistent or not. They are *persistent* if repeated queries to the same domain element always return the same answer. In general, the case of persistent errors is more difficult, since non-persistent errors can yield extra information, and can always be made persistent simply by caching and using the first answer for each domain point queried.

Sakakibara defines one model of non-persistent errors, in which each answer to a query may be wrong with some probability, and repeated queries constitute independent events [12]. He gives a general technique of repeating each query sufficiently often to establish the correct answer with high probability. This yields a uniform transformation of existing query algorithms. This could be a reasonable model of a situation in which the answers to queries were being transmitted through a medium subject to random independent errors; then the technique of repeating the query is eminently sensible.

A related model is considered by Dean et al. for the case of a robot learning a finite-state map of its environment using faulty sensors and reliable effectors [7]. In this model, observation errors are taken as independent as long as there is a nonempty action sequence separating them. This means that there is no simple way to "repeat the same query", since a nonempty action sequence may take the robot to another state, and no reset operation is available. A polynomial-time learning algorithm is given for the situation in which the environment has a known distinguishing sequence. It achieves exact identification with high probability.

To deal with the more difficult case of persistent errors in membership queries, the idea of "repeating the query" is insufficient, and the success becomes dependent on the error-correcting properties of groups of "related" queries. In an explicit and very interesting application of the ideas of error-correcting algorithms, Ron and Rubinfeld use the criterion of PAC-identification with respect to the uniform distribution, and give a polynomial-time randomized algorithm using membership queries to learn DFA's with high rates of random persistent errors in the answers to the membership queries [11].

Algorithms that use membership queries to estimate probabilities (in the spirit of the statistical queries defined by Kearns [9]) are generally not too sensitive to small rates of random persistent errors in the answers to queries. For example, Goldman,

Kearns and Schapire give polynomial-time algorithms for exactly learning read-once majority formulas and read-once positive NAND formulas of depth $O(\log n)$ with high probability using membership queries with high rates of persistent random noise or modest rates of persistent malicious noise [8]. As another example, the algorithm of Kushilevitz and Mansour that uses membership queries and exactly learns logarithmic depth decision trees with high probability in polynomial time seems likely to be robust under nontrivial rates of persistent random noise in the answers to queries [10].

However, other algorithms depend more strongly on the correctness of the answers to individual queries; in these cases, the existence of an error-tolerant algorithm for the problem is in question. This is particularly true of learning algorithms in the equivalence and membership query model, where often the class of concepts being learned is known not to be learnable in polynomial time using equivalence queries only.

Our goal in this paper is to introduce a variant of the equivalence and membership query model in which the effects of errors in the answers to membership queries can be isolated for study. For this reason, we postulate a model in which equivalence queries remain correct (so that exact identification is still possible, if only by enumeration of hypotheses), and there may be errors in the answers to membership queries. Another motivation for this separation is that we may imagine the membership queries to be answered by a (less than omniscient) teacher, whereas the equivalence queries correspond to comparisons of the hypothesis with real-world results.

The first such model was introduced by Angluin and Slonim: equivalence queries are assumed to be answered correctly, while membership queries are either answered correctly or with "I don't know" and the answers are persistent. The "I don't know" answers are determined by independent coin flips the first time each query is made [4]. They give a polynomial-time algorithm to learn monotone DNF formulas with high probability in this setting. They also show that a variant of this algorithm can deal with one-sided errors, assuming that no negative point is classified as positive.

In this paper we consider malicious, persistent errors in the answers to membership queries. There is no element of randomness in the choice of which membership queries are answered incorrectly — the only limitation is a parameter L, which is an upper bound on the number plus the sum of lengths of elements of the domain whose membership queries are answered incorrectly. We assume that the answers to equivalence queries are correct, but choice of which (correct) counterexample to give is under the control of the same adversary choosing which membership queries to answer incorrectly. We require exact identification of the target concept, but permit

polynomial time dependence on the usual parameters and L.

2 Preliminaries

2.1 Concepts and Concept Classes

Our definitions for concepts and concept classes are a bit non-standard. We have explicitly introduced the domains of concepts in order to try to unify the treatment of fixed-length and variable-length domains. We take Σ and Γ to be two finite alphabets. Examples are represented by finite strings over Σ and concepts are represented by finite strings over Γ.

A *concept* consists of a pair (X, f), where $X \subseteq \Sigma^*$, and f maps X to $\{0,1\}$. X is the *domain* of the concept. The *positive examples* of (X, f) are those $w \in X$ such that $f(w) = 1$, and the *negative examples* of (X, f) are those $w \in X$ such that $f(w) = 0$. Note that strings not in the domain of the concept are neither positive nor negative examples of it.

A *concept class* is a triple (R, Dom, μ), where R is a subset of Γ^*, Dom is a map from R to subsets of Σ^*, and for each $r \in R$, $\mu(r)$ is a function from $Dom(r)$ to $\{0,1\}$. R is the set of legal representations of concepts. For each $r \in R$, the *concept represented by* r is $(Dom(r), \mu(r))$.

A concept (X, f) is *represented by* a concept class (R, Dom, μ) if and only if for some $r \in R$, (X, f) is the concept represented by r. The *size* of a concept (X, f) represented by (R, Dom, μ) is defined to be the length of the shortest string $r \in R$ such that r represents (X, f). The size of (X, f) is denoted by $|(X, f)|$; note that it depends on the concept class chosen.

The concept classes we consider in this paper are the boolean formulas and syntactically restricted subclasses of them, boolean decision trees, and deterministic finite acceptors (DFA's). The representations are more or less standard, except each concept representation specifies the relevant domain. For DFA's, the domain of every concept is the set Σ^* itself. For boolean formulas and decision trees, we assume that $\Sigma = \{0,1\}$, and each concept representation specifies a domain of the form $\{0,1\}^n$.

2.2 Exceptions

For each finite set S of strings from Σ^*, we define its *table-size*, denoted $||S||$, as the sum of the lengths of the strings in S and the number of strings in S. Note that $||S|| = 0$ if and only if $S = \varnothing$. For a concept (X, f) and a finite set $S \subseteq X$, we define *the concept* (X, f) *with exceptions* S, denoted $xcpt((X, f), S)$, as the concept (X, f') where $f'(w) = f(w)$ for strings in $X - S$, and $f'(w) = 1 - f(w)$ for strings in S. (Thus f and f' have the same domain, and are equal except on the set of strings S, which is a subset of their common domain.) It is useful to note that S is partitioned by (X, f) into a set of *positive exceptions* S_+ and a set of *negative exceptions* S_-, defined by $S_+ \stackrel{\text{def}}{=} S - \{w \in X \mid f(w) = 1\}$ and $S_- \stackrel{\text{def}}{=} S \cap \{w \in X \mid f(w) = 1\}$.

A concept class (R, Dom, μ) is *closed under finite exceptions* provided that for every concept (X, f) represented by (R, Dom, μ) and every finite set $S \subseteq X$, the concept $xcpt((X, f), S)$ is also represented by (R, Dom, μ). If, in addition, there is a fixed polynomial of two arguments such that the concept $xcpt((X, f), S)$ is of size bounded by this polynomial in the size of (X, f) and $||S||$, we say that (R, Dom, μ) is *polynomially closed under finite exceptions.*

This definition differs from the similar one given in [5] because we do not require that there exists a polynomial-time algorithm that given a concept and a list of exceptions produces the new concept. However, for the classes that we consider there are such algorithms.

We define a natural operation of adding finite exception tables to a class of concepts to produce another class of concepts that "embeds" the first and is polynomially closed under finite exceptions.

We assume $\Sigma \subseteq \Gamma$ and $|\Gamma| \geq 2$. We define a simple encoding e that takes a string r from Γ^* and a finite set of strings $S \subseteq \Sigma^*$ and produces a string r' in Γ^* from which r and the elements of S can easily be recovered, and is such that $|r'| = 2(1 + |r| + ||S||)$. The details of the encoding are as follows.

Assume that 0 and 1 are distinct symbols in Γ. We define

$$e_b(b_1 b_2 \ldots b_j) \stackrel{\text{def}}{=} bbbb_1 bb_2 \ldots bb_j,$$

for $b \in \{0, 1\}$ and $b_1, b_2, \ldots, b_j \in \Gamma$. Note that $|e_b(w)| = 2(1 + |w|)$ for every string $w \in \Gamma^*$. We then define the encoding of r and S as

$$r' = e(r, S) \stackrel{\text{def}}{=} e_0(r) e_1(s_1) e_0(s_2) \ldots e_{k \bmod 2}(s_k),$$

where $s_1, s_2, \ldots, s_k$ are the strings in S.

Given a concept class (R, Dom, μ), we define the *class obtained from it by adding exception tables* as (R', Dom', μ'), where R' is the set of all strings of the form $e(r, S)$ such that $r \in R$ and S is a finite subset of $Dom(r)$, and for each $r' \in R'$, the concept represented by $r' = e(r, S)$ is the concept represented by r with exceptions S, that is, $(Dom'(r'), \mu'(r')) = xcpt((Dom(r), \mu(r)), S)$.

For example, adding exception tables to the monotone DNF formulas produces a concept class which we term *monotone DNF formulas with finite exceptions.* More detailed discussion of classes obtained by adding exception tables and of polynomial closure under finite exceptions can be found in Section 5.

2.3 Queries

In our setting, the goal of a learning algorithm is exact identification of a target concept $(Dom(r), \mu(r))$ chosen from a known concept class (R, Dom, μ). We assume that the domain $Dom(r)$ of the target concept is also known to the learning algorithm (which for boolean formulas and decision trees means n, the number of attributes, is known).

Information about the target concept is available to the algorithm as the answers to two types of queries: equivalence queries and malicious membership queries.

In an equivalence query, the algorithm gives as input a concept $r' \in R$ with the same domain as the target, and the answer depends on whether $\mu(r) = \mu(r')$. If so, the answer is "yes", and the learning algorithm has succeeded in its goal of exact identification of the target concept. Otherwise, the answer is a *counterexample*, any string $w \in Dom(r)$ on which the functions $\mu(r)$ and $\mu(r')$ differ. We denote an equivalence query on a hypothesis h by $\mathrm{EQ}(h)$.

A *label* for a counterexample $v = \mathrm{EQ}(r')$ is $h^*(v)$ or, equivalently, $(1 - h(v))$, where h^* is the target concept and $h = \mu(r')$ is the concept that equivalence query was made with. The counterexamples that have label 1 are called *positive counterexamples* and the ones that have label 0 are called *negative counterexamples.*

In a malicious membership query, the learning algorithm gives as input a string $w \in Dom(r)$, and the answer is either 1 or 0. If the answer is equal to the value of $\mu(r)$ on w, then the answer is *correct*, otherwise it is an *error.* We denote a malicious

membership query about a string x by MMQ(x).

Note that the learning algorithm has no *a priori* way to tell if an answer to a malicious membership query is correct or an error. However, the answers are restricted as follows.

1. They are *persistent*, that is, different queries with the same input string w receive the same answer. Queries that are not persistent may reveal some information, namely, if one query on a string w returns a different answer than another query on the same string, the algorithm knows that one of them is wrong. Every algorithm designed to work with persistent queries can be made to work with non-persistent ones by caching the queries and always using the first answer for subsequent queries.

2. In addition, we bound the "amount of lying" permitted in answers to malicious membership queries. One natural quantity to bound would be the number of different strings whose malicious membership queries can be answered incorrectly, and this works well in fixed-length domains. However, in variable-length domains, we wish to account for the lengths of the strings lied about as well as their number.

   Therefore, in general the algorithm is given a bound L on the table-size, $||S||$, of the set S of strings whose malicious membership queries are answered incorrectly during a single run. In the case of a fixed-length domain, $\{0,1\}^*$, we may instead give a bound ℓ on the number of different strings whose MMQ's are answered incorrectly. Note that $L = \ell(n+1)$ is a bound on the table-size in this case.

   Note that when $L = 0$ or $\ell = 0$ there can be no errors in the answers to MMQ's and we have the usual model of membership queries as a special case; for emphasis we will sometimes refer to these as *(vanilla) membership queries* in this paper. We denote a (vanilla) membership query about a string x by MQ(x).

We assume that an on-line adversary controls the choice of counterexamples in answers to equivalence queries and the choice of which elements of the domain will be answered with errors in malicious membership queries.

An algorithm *exactly learns* (R, Dom, μ) *in polynomial time using equivalence and malicious membership queries* if and only if there exists a polynomial $p(s, n, L)$ such that for every $r \in R$, every nonnegative integer L and every on-line adversary answering equivalence and malicious membership queries about r, at every point in the run

the algorithm's running time is bounded above by $p(s, n, L)$, where s is the size of the target concept, n is the maximum length of any counterexample given so far, L is a bound on the table-size of the strings for which MMQ's are answered incorrectly, and the algorithm eventually halts and outputs a concept $r' \in R$ with the same domain as r and such that $\mu('r) = \mu(r)$.

The definition is extended in the usual ways to cover randomized learning algorithms and their expected running times, and also extended equivalence queries, in which the inputs to equivalence queries and the final result of the algorithm are allowed to come from a concept class different from (usually larger than) the concept class from which the target is drawn.

2.4 Monotone DNF Formulas

We assume a set of propositional variables V and denote its elements by $x_1, x_2, \ldots, x_n$, where n is the cardinality of V. A monotone DNF formula over V is a DNF formula over V where no literal is negated. The domain of such a formula is $\{0,1\}^n$. For example, for $n = 20$,

$$x_1x_4 \vee x_2x_{17}x_3 \vee x_9x_5x_{12}x_3 \vee x_8$$

is a monotone DNF formula (with domain $\{0,1\}^{20}$) and a possible target concept in Section 3. Note that there is an efficient algorithm to minimize the number of terms of a monotone DNF formula. We will assume that the target formula h^* has been minimized. For a minimized monotone DNF formula f, let $m(f)$ denote the number of terms in f. We will, however, just write m when the formula it refers to is clear from the context. In the above example, $m = 4$.

We view the domain $\{0,1\}^n$ of monotone DNF formulas (with or without exceptions) as a lattice, with componentwise "or" and "and" as the lattice operations. The top element is the vector of all 1's, and the bottom element is the vector of all 0's. The elements are partially ordered by $\leq$, where $v \leq w$ if and only if $v[i] \leq w[i]$ for all $1 \leq i \leq n$. Often we refer to the examples as points of a hypercube $\{0,1\}^n$. For a point v, all points w such that $w \leq v$ are called the *descendants* of v. Those descendants that can be obtained by setting exactly one coordinate of v from a 1 to a 0 are called the *children* of v. The *ancestors* and the *parents* are defined similarly.

For convenience, we use a representation of monotone DNF formulas in which each term is represented by the minimum vector in the ordering $\leq$ that satisfies the term. Thus, vector 10011 (where $n = 5$) denotes the term $x_1x_4x_5$. In this representation, if

h is a monotone DNF formula and v is a vector in the sample space, v satisfies h if and only if for some term t of h, $t \leq v$.

For any n-argument boolean function f, we call point x a *local minimum point* of f if $f(x) = 1$ but for every child y of x in the lattice, $f(y) = 0$. The local minimum points of a minimized DNF formula represent its terms in our representation.

For two n-argument boolean functions f_1 and f_2 we define the set $Err(f_1, f_2)$ to be the set of points where they differ. I.e., $Err(f_1, f_2) = \{x \mid f_1(x) \neq f_2(x)\}$. The cardinality of $Err(f_1, f_2)$ is called the distance between f_1 and f_2 and is denoted by $d(f_1, f_2)$.

3 Learning Monotone DNF Formulas With Malicious Membership Queries

In this section we present an algorithm that uses equivalence and malicious membership queries to learn monotone DNF formulas. The key idea is to depend on equivalence queries as much as possible, since they are correct.

The algorithm keeps track of all the counterexamples and their labels received through equivalence queries and consults them first, before asking a membership query. The pairs of counterexamples and their labels are kept in a set named *CounterExamples*. Obviously, for a positive counterexample v, if $x \geq v$ then it is not worth making a membership query about x; it can only be a positive point. Similarly, for a negative counterexample v, if $x \leq v$ then x has to be a negative point of the target formula. For this reason we define a subroutine CHECKEDMQ and use it instead of a membership query. The subroutine is given in Figure 1.

As in [2] and [4], our algorithm also uses a subroutine REDUCE in order to move down in the lattice from a positive counterexample. All the membership queries are done using the subroutine CHECKEDMQ, which possibly lets the algorithm avoid some incorrect answers. The subroutine REDUCE is given in Figure 2.

The algorithm for exactly identifying monotone DNF formulas using equivalence queries and malicious membership queries is given in Figure 3.

The algorithm is based on a few simple ideas. A positive counterexample is reduced to a point that is added as a term to the existing hypothesis h, which is a

```
CheckedMQ(x)
{
    If (∃(v,1) ∈ CounterExamples s.t. x ≥ v)
        Return 1
    If (∃(v,0) ∈ CounterExamples s.t. x ≤ v)
        Return 0
    Return MMQ(x)
}
```

Figure 1: Subroutine CheckedMQ

```
Reduce(v)
{
    For (each child w of v)
        If (CheckedMQ(w) == 1)
            Return Reduce(w)
    Return v
}
```

Figure 2: Subroutine Reduce

monotone DNF. That is, the new hypothesis will classify the latest counterexample and possibly some other points as positive.

Negative counterexamples are used to detect inconsistencies between membership and equivalence queries. They show that there have been errors in membership queries that have caused wrong terms to be added to the hypothesis. The algorithm reacts by removing all the terms that are inconsistent with the latest counterexample. These are the terms that have the negative counterexample above them. A term can be removed only when there is a negative counterexample above it.

4 Analysis of LearnMonDNF

Theorem 1 LearnMonDNF *learns the class of monotone DNF formulas in polynomial time using equivalence and malicious membership queries.*

```
LEARNMONDNF()
{
    CounterExamples = ∅
    h = "the empty formula"
    While ((v = EQ(h)) ≠ "yes")
    {
        Add (v, 1 − h(v)) to CounterExamples
        If (h(v) == 0)
        {
            w = REDUCE(v)
            Add term w to h
        }
        Else
            For (each term t of h)
                If (t(v) == 1)
                    Delete term t from h
    }
    Output h
}
```

Figure 3: The algorithm for learning monotone DNF formulas

We need a definition and a simple lemma before proving the theorem.

Let h^* be a monotone boolean function on $\{0,1\}^n$, and let h' be an arbitrary boolean function on $\{0,1\}^n$. Let C be any subset of $\{0,1\}^n$. The *monotone correction of h' with h^* on C*, denoted $mc(h', h^*, C)$, is the boolean function h'' defined for each string $x \in \{0,1\}^n$ as follows.

$$h''(x) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if there exists } y \in C \text{ such that } y \leq x \text{ and } h^*(y) = 1, \\ 0 & \text{if there exists } y \in C \text{ such that } x \leq y \text{ and } h^*(y) = 0, \\ h'(x) & \text{otherwise.} \end{cases}$$

Note that since h^* is monotone, the first two cases above cannot hold simultaneously. It is also clear that if the value of $h''(x)$ is determined by one of the first two cases, $h''(x) = h^*(x)$. We prove a simple monotonicity property of the monotone correction operation.

Lemma 1 *Suppose h^* is a monotone boolean function and h' is an arbitrary boolean function on $\{0,1\}^n$. Let $C_1 \subseteq C_2$ be two subsets of $\{0,1\}^n$. Let $h_1 = mc(h', h^*, C_1)$ and $h_2 = mc(h', h^*, C_2)$. Then the set of points on which h_2 and h^* differ is contained in the set of points on which h_1 and h^* differ. That is, $Err(h_2, h^*) \subseteq Err(h_1, h^*)$.*

Proof: Let x be an arbitrary point on which $h_2(x) \neq h^*(x)$. Then it must be that $h_2(x) = h'(x)$ and there does not exist any point $y \in C_2$ such that $x \leq y$ and $h^*(x) = 0$ or $y \leq x$ and $h_*(x) = 1$. Since C_1 is contained in C_2, there is no point $y \in C_1$ such that $x \leq y$ and $h^*(x) = 0$ or such that $y \leq x$ and $h_*(x) = 1$. Thus, $h_1(x) = h'(x)$ and $h_1(x) \neq h^*(x)$. Consequently, $Err(h_2, h^*) \subseteq Err(h_1, h^*)$. ■

Now we start the proof of Theorem 1.

Proof: Let h^* denote the target concept, an arbitrary monotone DNF formula over $\{0,1\}^n$ with m terms. Let ℓ be a bound on the number of strings whose MMQ's are answered incorrectly. Because equivalence queries are answered correctly, if the algorithm ever halts, the hypothesis output is correct, so we may focus on proving a polynomial bound on the running time.

Since LEARNMONDNF is deterministic and the target concept h^* is fixed, we may assume that the adversary chooses in advance how to answer all the queries, that is, chooses a sequence $y_1, y_2, \ldots$ of counterexamples to equivalence queries and a set S of strings on which to answer MMQ's incorrectly. Note that $|S| \leq \ell$.

In turn, these choices determine a particular computation of LEARNMONDNF which we now focus on. It suffices to bound the length of this computation. In this computation the answers to MMQ's agree with the boolean function $h_0 = xcpt(h^*, S)$. Also, if CHECKEDMQ is called at step t of this computation on the string x, the answer agrees with the boolean function $mc(h_0, h^*, C)$, where C is set of counterexamples received up to step t.

The set *CounterExamples* only changes when a new counterexample is received. Therefore, the successive distinct sets of counterexamples in this computation can be denoted by $C_0, C_1, \ldots$, where $C_0 = \varnothing$ and $C_i = C_{i-1} \cup \{y_i\}$, for $i = 1, 2, \ldots$. If we also define

$$h_i = mc(h_0, h^*, C_i)$$

for $i = 1, 2, \ldots$, then CHECKEDMQ answers according to h_0 until the first counterexample is received, then according to h_1 until the second counterexample is received, and so on.

Clearly, since h_0 disagrees with h^* on at most ℓ strings, $d(h_0, h^*) \leq \ell$. Since the sets $C_0, \ldots$ are monotonically nondecreasing, Lemma 1 shows that $Err(h_i, h^*) \subseteq Err(h_{i-1}, h^*)$ for $i = 1, 2, \ldots$.

We say that a counterexample y_i *corrects a positive error* at point x, if $h_{i-1}(x) = 1$ but $h_i(x) = h^*(x) = 0$. We say that a counterexample y_i *corrects a negative error* at point x, if $h_{i-1}(x) = 0$ but $h_i(x) = h^*(x) = 1$. Note that from the construction of CHECKEDMQ it follows that positive errors can be corrected only by negative counterexamples and negative errors can be corrected only by positive counterexamples. Let there be ℓ_p positive and ℓ_n negative errors corrected in the whole computation. Of course, $\ell_p + \ell_n \leq \ell$.

Claim 1 *If* REDUCE *is called after counterexample* y_i *and before counterexample* y_{i+1}, *it returns a local minimum point of* h_i.

Proof: After y_i is added to *CounterExamples*, CHECKEDMQ answers according to h_i. The claim follows from the construction of REDUCE. ■

Claim 2 *Condition preserved: at the* $(i+1)$*th equivalence query* EQ(h), *each term of* h *is a positive point of* h_i.

Proof: We prove the claim by induction.

> **Basis:** The first EQ is made on an empty formula. Thus, the claim is vacuously true.
>
> **Induction step:** Suppose the claim is true up to the ith EQ. Let h' be the hypothesis h at the ith EQ and h'' be the hypothesis h at the $(i+1)$th EQ. There are two cases to consider.
>
> > **Case 1:** y_i is a positive counterexample. Then $h_i(x) = 1$ if and only if $h_{i-1}(x) = 1$ or $x \geq y_i$. Let t be the term returned by REDUCE(y_i). Then $h'' = h' \vee t$. Let t'' be a term in h''. Then either t'' is a term of h' or $t'' = t$. If t'' is a term of h' then $h_{i-1}(t'') = 1$ by the inductive assumption and therefore $h_i(t'') = 1$. If $t'' = t$ then $h_i(t'') = 1$ since t was returned by REDUCE(y_i) which used CHECKEDMQ, which answered according to h_i.

Case 2: y_i is a negative counterexample. Then $h_i(x) = 1$ if and only if $h_{i-1}(x) = 1$ and $x \not\leq y_i$. Let t'' be a term in h'', which consists of all those terms t' of h' such that $t' \not\leq y_i$. Therefore, $t'' \not\leq y_i$ and by the inductive assumption $h_{i-1}(t'') = 1$. It follows that $h_i(t'') = 1$. ■

Claim 3 *Once a term x is deleted from hypothesis h, it can never reappear in it.*

Proof: Since x was deleted, there must have been a negative counterexample y_i such that $y_i \geq x$. But then $(y_i, 0)$ belongs to *CounterExamples* and CHECKEDMQ(x) can never return 1 again, which is necessary for x to be added to h. ■

We divide the run of the algorithm into non-overlapping *stages*. A new stage begins either at the beginning of the run or with a new negative counterexample. Thus with each new stage *CounterExamples* contains one more negative counterexample and some (possibly none) new positive counterexamples. The following claim establishes that the distance $d(h_i, h^*)$ decreases with every new stage.

Claim 4 *Every negative counterexample corrects at least one error. More formally, if y_i is a negative counterexample, then there exists $x \in \{0,1\}^n$ such that $h_{i-1}(x) = 1$ and $h_i(x) = h^*(x) = 0$.*

Proof: Let y_i be a negative counterexample returned by EQ(h). Hence $h(y_i) = 1$, and there is some term $x \leq y_i$ in h. By Claim 2, $h_{i-1}(x) = 1$.

Since $h^*(y_i) = 0$ and $y_i \geq x$ it follows that $h^*(x) = 0$. By the definition of h_i it follows that $h_i(x) = 0$. ■

From Claim 4 it follows that there are at most ℓ_p negative counterexamples. Hence there are at most $\ell_p + 1$ stages in the run of the algorithm.

We divide each stage of the algorithm into non-overlapping *substages*. A substage begins either at the beginning of a stage or with a new positive counterexample that corrects an error. Obviously there can be no more than ℓ_n positive counterexamples that correct errors and hence no more than $\ell_p + \ell_n + 1$ substages in the whole run of the algorithm. The distance $d(h_i, h^*)$ decreases with every new substage. If, however, functions h_i and h_j belong to the same substage, they are equivalent and their local minima are the same. This allows us to bound the total number of positive counterexamples.

Claim 5 *Every new positive counterexample is reduced to a local minimum point of $h_0, h_1, \ldots$ that has not been found earlier.*

Proof: Let v be a positive counterexample that REDUCE is started with. Let t be the point REDUCE(v) returns. Assume, by way of contradiction, that t has already been found before. From Claim 3 it follows that t is a term in h. Since $v \geq t$, it follows that $h(v) = 1$. This is a contradiction to the assumption that v is a positive counterexample. ■

We denote the set of local minimum points of a boolean function f by $Lmp(f)$. We bound the total number of different local minima of the functions $h_0, h_1, \ldots$.

Lemma 2 *Let f and f' be n-argument boolean functions such that $Err(f, f') = \{x\}$. Then*

(a) If $f'(x) = 1$ then $|Lmp(f') - Lmp(f)| \leq 1$.

(b) If $f'(x) = 0$ then $|Lmp(f') - Lmp(f)| \leq n$.

Proof:

(a) The only point that can be a local minimum of f' and is not a local minimum of f, is x itself. The claim follows immediately.

(b) Any point which is a local minimum of f' but not of f is a parent of x. Since x has at most n parents, the claim follows. ■

Corollary 1 *Let f and f' be n-argument boolean functions such that $Err(f, f')$ contains d_p positive points of f' and d_n negative points of f'. Then*

$$|Lmp(f') - Lmp(f)| \leq nd_n + d_p.$$

Corollary 2 *Let $g_0, g_1, \ldots, g_r$ be the subsequence of $h_0, h_1, \ldots$, such that each g_i is the first of all the h_j's in its substage. Let $Err(h^*, g_{i-1}) - Err(h^*, g_i)$ contain $\ell_{p,i-1}$ positive and $\ell_{n,i-1}$ negative points of h^* for all $i = 1, 2, \ldots, r$. Let $Err(h^*, g_r)$ contain $\ell_{p,r}$ positive and $\ell_{n,r}$ negative points of h^*. Then the total number of different local minima of functions $g_0, g_1, \ldots, g_r, h^*$ is bounded above by $m + n\sum_{i=0}^{r} \ell_{n,i} + \sum_{i=0}^{r} \ell_{p,i}$.*

Proof: Note that $g_0, g_1, \ldots, g_r$ are the different functions in $h_0, h_1, \ldots$, and that CHECKEDMQ first answers according to g_0, then according to g_1 and so on. Obviously, $Err(h^*, g_i) \subseteq Err(h^*, g_{i-1})$ and $Err(g_{i-1}, g_i) = Err(h^*, g_{i-1}) - Err(h^*, g_i)$ for all $i = 1, 2, \ldots, r$. Also note that for each $i = 0, 1, \ldots, r-1$, one of $\ell_{p,i}$ and $\ell_{n,i}$ is 0, but $\ell_{p,r}$ and $\ell_{n,r}$ may both be positive.

We want to find $|\bigcup_{i=0}^{r} Lmp(g_i) \cup Lmp(h^*)|$, knowing that $|Lmp(h^*)| = m$. Since

$$\bigcup_{i=0}^{r} Lmp(g_i) \cup Lmp(h^*) \subseteq Lmp(h^*) \cup (Lmp(g_r) - Lmp(h^*)) \cup \bigcup_{i=0}^{r-1} (Lmp(g_i) - Lmp(g_{i+1})),$$

from Corollary 1 it follows that

$$|\bigcup_{i=0}^{r} Lmp(g_i) \cup Lmp(h^*)| \leq |Lmp(h^*)| + (n\ell_{n,r} + \ell_{p,r}) + \sum_{i=0}^{r-1} (n\ell_{n,i} + \ell_{p,i})$$

and the bound follows. ■

Since each error can be corrected at most once, it follows that $\sum_{i=0}^{r} \ell_{n,i} \leq \ell_n$ and $\sum_{i=0}^{r} \ell_{p,i} \leq \ell_p$. Hence the total number of the local minima and the total number of positive counterexamples that can be found in a computation is bounded by $m + n\ell_n + \ell_p$. The number of negative counterexamples in a complete run is bounded by the number of positive errors. The total number of counterexamples is therefore bounded by $m + \ell_n n + \ell_p + \ell_p \leq m + \ell(n+1) = O(m + \ell n)$.

We now count the number of membership queries in a complete run of the algorithm. Each positive counterexample v may cause at most $n(n+1)/2$ membership queries, before REDUCE(v) returns. Therefore there can be at most $O(mn^2 + \ell n^3)$ membership queries in a complete run of the algorithm.

It is also clear that the running time of the algorithm is polynomial in m, n and ℓ. This concludes the proof of Theorem 1. ■

5 Finite Exceptions

The relevant definitions for this section were introduced in Section 2.2. Here we give a few lemmas and examples.

Example 1 The class of regular languages represented by DFA's is polynomially closed under finite exceptions. In [5] there is given an algorithm that takes as input

a DFA M and an exception set S and produces a new DFA for $xcpt(M, S)$. Its size is polynomial in the size of M and S.

Example 2 Another example of a class that is polynomially closed under finite exceptions is the class of boolean decision trees. This result is taken from [5] but since the construction is not given there, we sketch it here.

Lemma 3 *The class of boolean decision trees is polynomially closed under finite exceptions.*

Proof: Let T be a decision tree on n variables. Let S be the exception set for T. We construct the decision tree for $xcpt(T, S)$ as follows. We treat each exception point $x \in S$ individually. First we walk down from the root of the original tree T to see where x fits in it. If it leads us to a leaf with depth n, i.e., if all variables are tested on this path, we just reverse the value of the leaf, because this path is for x only. However, if we find ourselves at a leaf with depth less than n, we have to add new internal nodes to the tree. Denote the value of this leaf by b. We then continue the path that led us to this leaf with a path in which all the remaining variables are tested. We end the path by a leaf with value $1 - b$. For each new internal node on the path, we make the other child (the one not on the path) a leaf, and give it the original value b. Thus, each counterexample adds at most n new internal nodes to the tree. The size of the new tree, measured as the number of internal nodes, is bounded by $|T| + n \times |S| = |T| + ||S||$. ■

Example 3 One more interesting example is the class of DNF formulas.

Lemma 4 *The class of DNF formulas is polynomially closed under finite exceptions.*

Proof: Let f be an m-term DNF formula over n variables and S be an exception set for it. Let S be partitioned in the sets of positive and negative exceptions (S_+ and S_-, respectively), as described in Section 2.2. We construct a DNF formula for $xcpt(f, S)$ from the formula $(f \wedge f_-) \vee f_+$, where f_- is a DNF formula which is true on all the points in its domain except the ones in S_-, and f_+ is a DNF formula which is true exactly on the points in S_+. The domain for all these formulas is $\{0,1\}^n$.

Obtaining f_+ is easy — straightforward disjunction of all the *terms* in S_+, where we make terms from points by substituting the respective variable for a 1 value of

a coordinate and its negation for a 0 value. Obtaining f_- is harder. First we make a decision tree corresponding to f_-. We put each point from S_- individually in the tree as a 0-valued leaf at the end of a path of length n. All the remaining leaves get value 1. Then for each leaf with value 1 we make a term that will go into f_- by following the path from this leaf to the root. Obviously f_- has at most $n \times |S_-|$ terms. Thus, after "multiplying" the terms out, the formula $(f \wedge f_-) \vee f_+$ will have at most $mn \times |S_-| + |S_+| \leq (mn+1) \times |S|$ terms. ■

Example 4 By duality it follows that the class of CNF formulas is polynomially closed under finite exceptions.

Note that stronger bounds on the size of the new formula can be obtained by using the result in [14]. We, however, chose to present a simpler argument. Also note that the size bound is insufficient for *strong polynomial closure under exception lists* as defined in [5].

Example 5 As our final example we show that any class that is obtained by adding exception tables to another class is polynomially closed under finite exceptions.

Lemma 5 *Let (R, Dom, μ) be any class of concepts. Then the concept class obtained from it by adding exception tables is polynomially closed under finite exceptions.*

Proof: Let (R', Dom', μ') be the class obtained from (R, Dom, μ) by adding exception tables, as defined in Section 2.2. Let (X', f') be any concept from (R', Dom', μ') and let $r' \in R'$ be a shortest representation of (X', f'). Then there exists a concept $r \in R$ and a finite set $S \subseteq Dom(r)$, such that $((Dom'(r'), \mu'(r')) = xcpt((Dom(r), \mu(r)), S)$ and $|r'| = 2(1 + |r| + ||S||)$. Let $S' \subseteq Dom'(r') = Dom(r)$ be any finite set. Let concept h'' be defined as $h'' \stackrel{\text{def}}{=} xcpt((Dom'(r'), \mu'(r')), S')$. It is easy to see that $h'' = xcpt((Dom(r), \mu(r)), S \triangle S')$ and thus h'' is represented by some $r'' \in R'$ with size $2(1 + |r| + ||S \triangle S'||) \leq 2(1 + |r| + ||S|| + ||S'||) = |r'| + 2||S'||$. ■

Corollary 3 *The class of monotone DNF formulas with finite exceptions is polynomially closed under finite exceptions.*

6 Learning Monotone DNF Formulas With Finite Exceptions

In this section we present an algorithm that learns the class of monotone DNF formulas with finite exceptions. The target concept is a boolean function on n variables $h^* \stackrel{\text{def}}{=} xcpt(h^*_M, S^*)$, where h^*_M is some monotone DNF formula and S^* is a set of exceptions for it. The domain of the target concept is $\{0,1\}^n$.

We assume that we have an upper bound on the cardinality of S^* and denote it by l, i.e., $|S^*| \leq l$. If this bound is not known, we can start out by assuming it to be any positive integer and doubling it whenever convergence is not achieved within the proper time bound, which will be given later. We assume that h^*_M is minimized and has m terms.

Like LEARNMONDNF our current algorithm also has a set *CounterExamples* that stores pairs of all counterexamples and their labels received through equivalence queries. The purpose of it is slightly different: it lets the algorithm conclude that some points cannot be classified by h^*_M alone, and, therefore, have to be included in the exception set.

The algorithm tries to find a suitable monotone DNF formula, which, coupled with a proper exception set, would give the target concept. The equivalence queries are made on a pair (h, S) of a monotone DNF formula h and a set of exceptions S. It focuses only on building h and makes S from whatever in the set *CounterExamples* is currently misclassified by h. It uses a simple subroutine GETEXCEPTIONS for building S. The subroutine is given in Figure 4.

```
GETEXCEPTIONS()
{
        S = ∅
        For (each (x, b) ∈ CounterExamples)
                If (h(x) ≠ b)
                        Add x to S
        Return S
}
```

Figure 4: Subroutine GETEXCEPTIONS

In order to classify the counterexamples received, the algorithm needs to evaluate the current function $xcpt(h, S)$. This is done by another very simple subroutine THEFUNCTION, given in Figure 5.

```
THEFUNCTION(x)
{
        If (x ∈ S)
                Return 1 − h(x)
        Else
                Return h(x)
}
```

Figure 5: Subroutine THEFUNCTION

As in [2], [4] and Section 3, our algorithm also uses a subroutine REDUCE to move down in the lattice from a positive counterexample. Its goal is to reduce the positive counterexample to some point which can be added as a term to the formula h. Then the new hypothesis would classify the counterexample and possibly some other points as positive. However, this may not always be possible. There can be overwhelming evidence that the candidate point is just a positive exception and need not be added to h. More precisely, if there are more than l negative counterexamples above a term of h, then they all have to be in the exception set, which is then too big. Therefore the current subroutine REDUCE is somewhat more complex and checks whether a point has enough evidence to be an undoubted exception point or not. The subroutine is given in Figure 6.

```
REDUCE(v)
{
    For (each child w of v)
        If ((MQ(w) == 1) && (|{y ≥ w | (y, 0) ∈ CounterExamples}| ≤ l))
            Return REDUCE(w)
    Return v
}
```

Figure 6: Subroutine REDUCE

The algorithm for learning monotone DNF formulas with at most l exceptions using equivalence queries and membership queries is given in Figure 7.

```
LearnMonDNFwithFX()
{
    S = CounterExamples = ∅
    h = "the empty formula"
    While ((v = EQ((h, S))) ≠ "yes")
    {
        Add (v, 1 − TheFunction(v)) to CounterExamples
        If (TheFunction(v) == 1)
            For (each term t of h)
                If (|{w ≥ t | (w, 0) ∈ CounterExamples}| > l)
                    Delete term t from h
        For (each (x, 1) ∈ CounterExamples)
            If ((h(x) == 0) && (|{y ≥ x | (y, 0) ∈ CounterExamples}| ≤ l))
            {
                w = Reduce(x)
                Add term w to h
            }
        S = GetExceptions()
    }
    Output (h, S)
}
```

Figure 7: The algorithm for learning monotone DNF formulas with finite exceptions

The algorithm is based on the following ideas. Each positive counterexample is reduced if possible to a new term to be added to the formula, as was explained above. In case this is not possible, the algorithm benefits anyway by storing it in the set *CounterExamples*.

Negative counterexamples imply that there are not as many positive points in the target concept as we thought. Sometimes more exception points are necessary for the hypothesis to be correct. Other times some terms have to be removed from the formula. Deleting a term happens only when there is enough evidence that a term is wrong, namely, more than l negative counterexamples above it.

7 Correctness and Complexity of the Algorithm

Theorem 2 LEARNMONDNFWITHFX *learns the class of monotone DNF formulas with exceptions in polynomial time using equivalence and (vanilla) membership queries.*

Proof: We begin the analysis with this simple claim.

Claim 6 *Once a term t is deleted from hypothesis h, it can never reappear in it.*

Proof: A term t can be deleted only if there are more than l negative counterexamples above it. To reappear, t must be returned by REDUCE. But every point returned by REDUCE must have at most l negative counterexamples above it at the time it is returned, so REDUCE cannot return t again. ■

The following lemma shows what points REDUCE can return.

Lemma 6 REDUCE *always returns either a local minimum of h^* or a parent of a positive exception in S^*.*

Proof: First note that REDUCE can only be called on points x such that $h^*(x) = 1$ and can only return points w such that $h^*(w) = 1$. Let w be a point returned by REDUCE. Assume w is not a local minimum point of h^*. Then for some child y of w, $h^*(y) = 1$ and the number of negative counterexamples above y is greater than l. Hence, y cannot be above any term t of h^*_M, since each term t can have at most l negative counterexamples above it. Therefore, y is a positive exception in S^*. ■

Now we are ready to bound the number of different points that can be returned by the subroutine REDUCE.

Claim 7 *The number of different points that* REDUCE *can return is at most $m + (n+1)l$.*

Proof: Let set S^* contain l_p positive exceptions and l_n negative exceptions, where $l_p + l_n \leq l$. h^*_M has m terms and thus m local minima. By Lemma 2, the number of local minima of h^* is at most $m + l_p + nl_n$. Each positive exception can have at most n parents which allows us to bound the number of parents of positive exceptions by nl_p. Therefore, by Lemma 6, the number of different points that REDUCE can return is at most $m + (n+1)l_p + nl_n \leq m + (n+1)l$. ■

All equivalence queries are asked about the current function $xcpt(h, S)$. Since S is constructed right before a new equivalence query from counterexamples that are misclassified by h alone, the argument of an equivalence query is always consistent with all the counterexamples seen to that point. Therefore the function $xcpt(h, S)$ is different for each equivalence query. This allows us to bound the number of equivalence queries.

Claim 8 *The number of equivalence queries before success is bounded by* $O(m^2n^2l^3)$.

Proof: We examine how $xcpt(h, S)$ changes. Either h itself changes, or h remains the same and S changes, namely, it contains exactly one point more, the most recent counterexample.

By Claim 6, each term of h can appear in it or disappear from it only once. Thus each possible term can induce at most 2 changes in formula h — first by appearing in it and then by disappearing. Thus, h can only change twice as many times as the number of terms that REDUCE can return. Therefore, by Claim 7, there can be at most $2(m + (n+1)l) + 1$ different functions h in a complete run of the algorithm.

We now count the number of times S can change while h remains the same. S grows larger by one with each new counterexample. It contains some (possibly none) points x such that $h(x) = 1$ and some (possibly none) points x such that $h(x) = 0$. We bound the number of each of these separately.

Each point $x \in S$ such that $h(x) = 1$ is above some term of h. No term can have more than l negative counterexamples above it. Therefore, the number of points $x \in S$ such that $h(x) = 1$ can be bounded by l times the bound $m + (n+1)l$ on the number of different terms of h, that is, by $ml + (n+1)l^2$.

Each point $x \in S$ such that $h(x) = 0$ is a positive counterexample. It is not above any term in h, which must be because x has more than l negative counterexamples above it. Otherwise the algorithm would have called REDUCE on x and added a new

term $t \leq x$ to h. If x has more than l negative counterexamples above it, then it cannot be above a term in h^*_M and thus has to be a positive exception in S^*. Hence we have a bound of l_p on the number of points $x \in S$ such that $h(x) = 0$.

Altogether, we can bound the cardinality of S by $|S| \leq ml + (n+1)l^2 + l_p \leq (m+1)l + (n+1)l^2$. While h stays the same, the number of possible different sets S is at most $(m+1)l + (n+1)l^2 + 1$.

Hence, the total number of equivalence queries in a complete run of the algorithm is bounded by $(2(m + (n+1)l) + 1) \times ((m+1)l + (n+1)l^2 + 1) = O(m^2n^2l^3)$. ■

We now count the total number of membership queries. Each positive counterexample v may cause at most $n(n+1)/2$ membership queries, before REDUCE(v) returns. Therefore there can be at most $O(m^2n^4l^3)$ membership queries in a complete run of the algorithm.

It is not difficult to see that the total running time of the algorithm is polynomial in n, m and l. This concludes the proof of Theorem 2. ■

8 Exceptions and Lies

In this section we focus on a relation between learning concepts with exceptions and learning with malicious membership queries. We give a generic algorithm transformation to learn in polynomial time using equivalence and malicious membership queries any class of concepts that is polynomially closed under finite exceptions and learnable in polynomial time with equivalence and membership queries.

Theorem 3 *Let H be a class of concepts that is polynomially closed under finite exceptions and learnable in polynomial time with equivalence and (vanilla) membership queries. Then H is learnable in polynomial time with equivalence and malicious membership queries.*

Proof: Let $H = (R, Dom, \mu)$ be a target class of concepts that is polynomially closed under finite exceptions. We assume that LEARN is an algorithm to learn H using equivalence (EQ) and (vanilla) membership queries (MQ) in time $p_A(s, n)$, for some polynomial p_A. Without loss of generality, p_A is non-decreasing in both arguments. We transform this algorithm into algorithm LEARNANYWAY which learns any concept

$h^* \in H$ using equivalence and malicious membership queries in time polynomial in $|h^*|$, n and the table-size L of the set of strings on which MMQ may lie.

As in Sections 3 and 6 the main idea is to keep track of all the counterexamples seen and to use them to avoid unnecessary membership queries. For this purpose we have the set *CounterExamples* again. As before it stores pairs of counterexamples and their labels. Now, before asking a membership query about string x, we scan *CounterExamples* to see whether it already contains x and a label for it. If x and the label are found, the algorithm knows the answer and does not make the query. (For some concept classes (e.g., monotone DNF formulas) it might be possible to infer the classification of x according to the target concept h^* even though x and its label are not contained in *CounterExamples*. However, this simple checking suffices for our algorithm and, what is more important, works in the general case.)

Another idea is to keep track of the answers received from membership queries, and to use them to conclude that MMQ has lied. For this purpose LEARNANYWAY has a set *MembershipAnswers*. This set stores pairs (x, b) for which MMQ was called on string x and returned answer b. After receiving a new counterexample from EQ, the algorithm stores it in *CounterExamples* and checks whether this counterexample is already contained in *MembershipAnswers*. If it is present in *MembershipAnswers* with the wrong label, the algorithm discards everything except the set *CounterExamples* and starts from scratch. If this is not the case, the algorithm continues the simulation of LEARN, which we now describe in detail.

The new algorithm simulates LEARN on the target concept by doing everything as it does, except for the following:

- Each membership query of LEARN, MQ(x), is replaced by a subroutine call NEWMQ(x). The subroutine is given in Figure 8.
- Each equivalence query of LEARN, x = EQ(h), as well as the output statement, **Output** h, is replaced by the block of code given in Figure 9.

Note that when the simulation is restarted, only the set *CounterExamples* reflects any work done so far. We now show that LEARNANYWAY is correct and runs in time polynomial in $|h^*|$, n and L. We partition the run of the algorithm into *stages*, where a stage begins with a new simulation of LEARN. First we show that a stage cannot last forever.

Claim 9 *Every stage ends in time polynomial in $|h^*|$, n and L.*

```
NEWMQ(x)
{
        If ((x, b) ∈ CounterExamples)
                Return b
        b = MMQ(x)
        Add (x, b) to MembershipAnswers
        Return b
}
```

Figure 8: Subroutine NEWMQ

Proof: Note that H is polynomially closed under finite exceptions, which means that there is a polynomial $p(\cdot,\cdot)$ such that for every concept $h \in H$ and every finite set $S \subseteq Dom(h)$ there exists a concept $h' \in H$ equal to $xcpt(h, S)$ such that size $|h'| \leq p(|h|, ||S||)$. Without loss of generality we can assume that p is non-decreasing in both arguments. We now prove that each stage ends in time bounded by $p_A(p(|h^*|, L), n)$, where we count only the time spent on LEARN operations, i.e., we do not count the simulation and bookkeeping overhead.

We prove this by contradiction. Assume that stage i goes over the limit. Let us look at the situation right after the number of simulated steps of LEARN exceeds the above time bound. Let S_i denote the set of strings the MMQ has lied about during this stage, up to the time bound. Let n denote the length of the longest counterexample received during this stage, up to the time bound.

None of the strings in S_i can belong to *CounterExamples*. Assume by way of contradiction otherwise. Let $x \in S_i$ be a string contained in *CounterExamples* with some label. S_i contains exactly the strings that the MMQ lied on in this stage and time bound, so there was a query MMQ(x). It must have happened before x was added to *CounterExamples*. But then at the moment it was added to *CounterExamples* it already belonged to *MembershipAnswers* and an inconsistency had to be found. The stage had to end.

Therefore, considering S_i as an exception set, all the information received by LEARN in this stage and within the given time bound is consistent with the concept $h' = xcpt(h^*, S_i) \in H$. LEARN either has to output h' in time bounded by $p_L(p(|h^*|, ||S_i||), n) \leq p_L(p(|h^*|, L), n)$, or it has to receive a counterexample $x \in S_i$. In the former case, LEARNANYWAY makes an equivalence query EQ(h') and receives a counterexample $x \in S_i$, since only counterexamples from S_i are possible at that

```
{
        x = EQ(h)
        If (x = "yes")
        {
                Output h
                Return
        }
        Add (x, 1 − h(x)) to CounterExamples
        If ((x, h(x)) ∈ MembershipAnswers)
        {
                MembershipAnswers = ∅
                Restart Simulation, retaining CounterExamples
        }
}
```

Figure 9: The block of code replacing "$x = \mathrm{EQ}(h)$" or "**Output** h"

point. In either case, an element of S_i is added to *CounterExamples* by the above time bound, which we showed above was impossible. This is a contradiction to the assumption that stage i goes over this bound. ■

What remains is to show that there can be only a small number of stages. That is, we do not restart the simulation too many times.

Claim 10 *There are at most $L+1$ stages in the run of the algorithm* LEARNANYWAY.

Proof: At the beginning of each stage (except the first one) the algorithm discovers a new string where the MMQ lies and from then on MMQ can never lie on this string again, because it is added to *CounterExamples*. To be more precise, MMQ does not get a chance to lie on this string because it is never asked about it again. Let S be the set of the strings that MMQ lies on. Since $|S| \leq ||S|| \leq L$, in stage $L+1$ the MMQ can lie on no strings (i.e., it is not asked queries about any of the strings where it may lie). Therefore LEARN has to converge to the target concept h^*. ■

The time spent on simulation and bookkeeping is clearly polynomial in $|h^*|$, n and L. Thus, LEARNANYWAY is a polynomial-time algorithm that uses equivalence and

malicious membership queries to learn the class of concepts $H = (R, Dom, \mu)$. This concludes the proof of Theorem 3. ■

As corollaries of Theorem 3 we have the following.

Corollary 4 *The class of regular languages, represented by DFA's, is learnable in polynomial time with equivalence and malicious membership queries.*

Proof: In [5] it was shown that this class of concepts is polynomially closed under finite exceptions. In [1] it was shown that it is learnable in polynomial time using membership and equivalence queries. ■

Corollary 5 *The class of boolean decision trees is learnable in polynomial time with extended equivalence and malicious membership queries.*

Proof: Lemma 3 shows that the class of boolean decision trees is polynomially closed under finite exceptions. In [6] it was shown that it is learnable in polynomial time using membership and extended equivalence queries. ■

Corollary 6 *The class of monotone DNF formulas with finite exceptions is learnable in polynomial time with equivalence and malicious membership queries.*

Proof: Corollary 3 shows that the class of monotone DNF formulas with exceptions is polynomially closed under finite exceptions. In Section 6 we gave an algorithm that learns this class in polynomial time with membership and equivalence queries. ■

Note that we can also learn the class of monotone DNF formulas without any exceptions with this generic algorithm, using extended equivalence and malicious membership queries, since it is just a subclass of the class that allows exceptions. However, the algorithm is much less efficient than the one described in Section 3.

9 Discussion and Open Problems

In Section 3 we show how to learn monotone DNF formulas in polynomial time with equivalence and malicious membership queries. Many open problems remain. One obvious question is: what can we say about lower bounds for identifying monotone DNF with equivalence and malicious membership queries? Can we prove something more than the trivial bound that there must be more membership queries than lies? Can we say something about lower bounds for other classes of concepts? Another open problem is finding polynomial-time algorithms with equivalence and malicious membership queries for other interesting concept classes.

In Section 6 we give a polynomial-time algorithm using equivalence and (vanilla) membership queries to learn the class of monotone DNF formulas with exceptions. Among the open problems regarding learning with exceptions are finding polynomial-time algorithms for other classes of concepts and proving lower bounds for any of the classes. Also, it is probably possible to improve the running time of the algorithm given in Section 6.

In Section 8 we show that there is a polynomial-time algorithm using equivalence and malicious membership queries for learning any concept class that is polynomially closed under finite exceptions and can be learned in polynomial time using equivalence and (vanilla) membership queries. Thus, learning with exceptions is not easier than learning with lies (modulo a polynomial-time reduction). An immediate question is whether it really is harder than learning with lies or they both are equally hard.

The generic method of Section 8 allows us to learn new classes with equivalence and malicious membership queries. These include DFA's and boolean decision trees. However, this result leaves the question open for other classes, polynomial-time learnable with equivalence and (vanilla) membership queries, such as read once formulas, that are not polynomially closed under finite exceptions. A start in this direction is made in [3], which gives a randomized polynomial-time algorithm to learn μ-DNF formulas with equivalence and malicious membership queries.

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comput.*, 75(2):87–106, November 1987.

[2] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[3] D. Angluin. Exact learning of μ-DNF formulas with malicious membership queries. Technical Report YALEU/DCS/TR-1020, Yale University Department of Computer Science, March 1994.

[4] D. Angluin and D. K. Slonim. Randomly fallible teachers: learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14(1):7–26, 1994.

[5] R. Board and L. Pitt. On the necessity of Occam algorithms. *Theoret. Comput. Sci.*, 100:157–184, 1992.

[6] N. Bshouty. Exact learning via the monotone theory. In *Proc. of the 34th Symposium on the Foundations of Comp. Sci.*, pages 302–311. IEEE Computer Society Press, Los Alamitos, CA, 1993.

[7] T. Dean, D. Angluin, K. Basye, S. Engelson, L. Kaelbling, E. Kokkevis, and O. Maron. Inferring finite automata with stochastic output functions and an application to map learning. In *Proceedings of AAAI-92*, pages 208–214. AAAI, 1992.

[8] S. A. Goldman, M. J. Kearns, and R. E. Schapire. Exact identification of read-once formulas using fixed points of amplification functions. *SIAM J. Comput.*, 22(4):705–726, 1993.

[9] M. Kearns. Efficient noise-tolerant learning from statistical queries. In *Proc. 25th Annu. ACM Sympos. Theory Comput.*, pages 392–401. ACM Press, New York, NY, 1993.

[10] E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. In *Proc. of the 23rd Symposium on Theory of Computing*, pages 455–464. ACM Press, New York, NY, 1991.

[11] D. Ron and R. Rubinfeld. Learning fallible finite state automata. In *Proc. 6th Annu. Workshop on Comput. Learning Theory*, pages 218–227. ACM Press, New York, NY, 1993.

[12] Y. Sakakibara. On learning from queries and counterexamples in the presence of noise. *Inform. Proc. Lett.*, 37(5):279–284, March 1991.

[13] L. G. Valiant. Learning disjunctions of conjunctions. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence, vol. 1*, pages 560–566, Los Angeles, California, 1985. International Joint Committee for Artificial Intelligence.

[14] Y. Zhuravlev and Y. Kogan. Realization of boolean functions with a small number of zeros by disjunctive normal forms, and related problems. *Soviet Math. Doklady*, 32:771–775, 1985.