Washington University in St. Louis
Washington University Open Scholarship

All Computer Science and Engineering Research | Computer Science and Engineering

Report Number: WUCS-93-23

1993-01-01

Asking Questions to Minimize Errors

Nader H. Bshouty, Sally A. Goldman, Thomas R. Hancock, and Sleiman Matar

A number of efficient learning algorithms achieve exact identification of an unknown function from some clas using membership and equivalence queries. Using a standard transformation such algorithms can easily be converted to on-line learning algorithms that use membership queries. Under such a transformation the number of equivalence queries made by the query algorithm directly corresponds to the number of mistakes made by the on-line algorithm. In this paper we consider several of the natural classes known to be learnable in this setting, and investigate the minimum number of equivalence queries with accompanying counterexamples (or equivalently the minimum number of... **Read complete abstract on page 2.**

Follow this and additional works at: https://openscholarship.wustl.edu/cse_research

 Part of the Computer Engineering Commons, and the Computer Sciences Commons

Recommended Citation

Bshouty, Nader H.; Goldman, Sally A.; Hancock, Thomas R.; and Matar, Sleiman, "Asking Questions to Minimize Errors" Report Number: WUCS-93-23 (1993). *All Computer Science and Engineering Research.*
https://openscholarship.wustl.edu/cse_research/311

Department of Computer Science & Engineering - Washington University in St. Louis
Campus Box 1045 - St. Louis, MO - 63130 - ph: (314) 935-6160.

This technical report is available at Washington University Open Scholarship: https://openscholarship.wustl.edu/cse_research/311

Asking Questions to Minimize Errors

Nader H. Bshouty, Sally A. Goldman, Thomas R. Hancock, and Sleiman Matar

Complete Abstract:

A number of efficient learning algorithms achieve exact identification of an unknown function from some clas using membership and equivalence queries. Using a standard transformation such algorithms can easily be converted to on-line learning algorithms that use membership queries. Under such a transformation the number of equivalence queries made by the query algorithm directly corresponds to the number of mistakes made by the on-line algorithm. In this paper we consider several of the natural classes known to be learnable in this setting, and investigate the minimum number of equivalence queries with accompanying counterexamples (or equivalently the minimum number of mistakes in the on-line model) that can be made by a learning algorithm that makes a polynomial nubmer of membership queries and uses polynomial computation time. We are able both to reduce the nubmer of equivalence queries used by the previous algorithms and often to prove matching lower bounds. As an example, consider the class of DNF formulas over n variables with at most k = O(logn) terms. Previously, the algorithm of Blum and Rudich [BR92] provided the best known upper bound of 2{raised to O9(k)}log(n) for the minimum number of equivalence queries needed for exact identification. We greatly improve on this upper bound showing that exactly k counterexamples are needed if the learning knows k a priori and exactly k+1 counterexamples are needed if the learner does not know k a priori. This exactly matches known lower bounds [BC92]. For many of our results we obtain a complete characterization of the tradeoff between the number of membership and equivalence queries needed for exact identification. The classes we consider here are monotone DNF formulas, Horn sentences, O(log(n))-term DNF formulas, read-k sat-j DNF formulas, read-once formulas over various baes, and deterministic finite automats.

Asking Questions to Minimize Errors

Nader H. Bshouty, Sally A. Goldman, Thomas R. Hancock and Sleiman Matar

WUCS-93-23

September 1993

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

Asking Questions to Minimize Errors

Nader H. Bshouty
Department of Computer Science
The University of Calgary
2500 University Drive N.W.
Calgary, Alberta, Canada T2N 1N4
bshouty@cpsc.ucalgary.ca

Sally A. Goldman
Department of Computer Science
Washington University
St. Louis, MO 63130
sg@cs.wustl.edu

Thomas R. Hancock
Siemens Corporate Research, Inc.
755 College Road East
Princeton, NJ 08540
hancock@learning.siemens.com

Sleiman Matar
Department of Computer Science
The University of Calgary
2500 University Drive N.W.
Calgary, Alberta, Canada T2N 1N4
sleiman@cpsc.ucalgary.ca

September 8, 1993

Abstract

A number of efficient learning algorithms achieve exact identification of an unknown function from some class using membership and equivalence queries. Using a standard transformation such algorithms can easily be converted to on-line learning algorithms that use membership queries. Under such a transformation the number of equivalence queries made by the query algorithm directly corresponds to the number of mistakes made by the on-line algorithm. In this paper we consider several of the natural classes known to be learnable in this setting, and investigate the minimum number of equivalence queries with accompanying counterexamples (or equivalently the minimum number of mistakes in the on-line model) that can be made by a learning algorithm that makes a polynomial number of membership queries and uses polynomial computation time. We are able both to reduce the number of equivalence queries used by the previous algorithms and often to prove matching lower bounds. As an example, consider the class of DNF formulas over n variables with at most $k = O(\log n)$ terms. Previously, the algorithm of Blum and Rudich [BR92] provided the best known upper bound of $2^{O(k)} \log n$ for the minimum number of equivalence queries needed for exact identification. We greatly improve on this upper bound showing that exactly k counterexamples are needed if the learner knows k a priori and exactly $k+1$ counterexamples are needed if the learner does not know k a priori. This exactly matches known lower bounds [BC92]. For many of our results we obtain a complete characterization of the tradeoff between the number of membership and equivalence queries needed for exact identification. The classes we consider here are monotone DNF formulas, Horn sentences, $O(\log n)$-term DNF formulas, read-k sat-j DNF formulas, read-once formulas over various bases, and deterministic finite automata.

1 Introduction

A very well studied formal learning model is the membership and equivalence query model developed by Angluin [Ang88]. In this model the learner's goal is to learn *exactly* how an unknown target function f, taken from some known representation class $\mathcal{C}$, classifies all instances from the domain. This goal is commonly referred to as *exact identification*. The learner has available two types of queries to find out about f: one is a membership query, in which the learner supplies an instance x from the domain and is told $f(x)$. The other query provided is an equivalence query in which the learner presents a candidate function h and either is told that $h \equiv f$ (in which case learning is complete), or else is given a counterexample x for which $h(x) \neq f(x)$. There is a very close relationship between this learning model and the on-line learning model [Lit88]. In the on-line learning model the learning session is divided into a set of trials where in each trial the learner is asked to make a prediction for some unknown instance x from the domain. After the prediction is made, the learner is told whether the prediction is correct and is then able to use polynomial time (and here a polynomial number of membership queries) before proceeding to the next trial. Using a standard transformation [Ang88, Lit88] algorithms that use membership and equivalence queries can easily be converted to on-line learning algorithms that use membership queries. Under such a transformation the number of counterexamples provided to the learner in response to the learner's equivalence queries directly correspond to the number of mistakes made by the on-line algorithm.

In the membership and equivalence query model a number of interesting polynomial time algorithms have been presented to learn target classes such as deterministic finite automata [Ang87b], Horn sentences [AFP90], read-once formulas [AHK89, BHH92a, BHH92b], k-term DNF formulas [BR92], etc. It is easily shown that membership queries alone are not sufficient for efficient learning of these classes, and Angluin has developed a technique of "approximate fingerprints" to show that equivalence queries alone are also not enough [Ang90]. (In both cases the arguments are information theoretic, and hold even when the computation time is unbounded.) Our research extends Angluin's results to establish tight bounds on how many equivalence queries are required for a number of these classes. Maass and Turán have also studied upper and lower bounds on the number of equivalence queries required for learning, both with and without membership queries [MT92]. However they do not restrict the learner to run in polynomial time, and they count only the total number of queries rather than the individual number of queries of each type.

Previous work generally makes the assumption that both types of queries have an equivalent cost to the learner (namely, constant cost). Thus there was no reason to favor one type of query over the other. Often in reality, one type of query is significantly less expensive to implement. In particular, we are interested in the learning problems in which membership queries are relatively inexpensive to perform (i.e. a simple experiment that can be run by the learner) whereas equivalence queries are expensive (i.e. require a "teacher"'s supervision to provide a counterexample). Furthermore, if you view the complexity of the learning algorithm under the on-line learning model, then reducing

the number of equivalence queries directly corresponds to minimizing the number of prediction mistakes.

The goal of our work is to establish tight bounds on how many equivalence queries are required when the learner is restricted to use a polynomial number of membership queries. Unless otherwise stated, in all upper bounds we restrict the learner to use polynomial time and learn with an algorithm whose output hypothesis comes from the concept class from which the target is selected. However, *all* of our lower bounds place no restrictions on the computation time of the learning algorithm and allow the learner to propose any hypothesis of its choice. In several cases we have obtained a complete characterization of the tradeoff between the number of membership and equivalence queries needed for exact identification. As an example of the type of results we have obtained, consider the problem of learning a formula from the class of DNF formulas over n variables with at most $k = O(\log n)$ terms. Previously, the algorithm of Blum and Rudich [BR92] provided the best known upper bound of $2^{O(k)} \log n$ for the minimum number of equivalence queries needed for exact identification. We greatly improve on this upper bound by giving an efficient algorithm that requires at most $k+1$ equivalence queries if k is unknown to the learner and at most k equivalence queries if the learner knows k in advance. This algorithm (as does Blum and Rudich's) uses equivalence queries from the class of arbitrary DNF formulas. Using a different technique we have designed an efficient algorithm that uses only k-term DNF formulas for hypotheses and has the same upper bound on the number of equivalence queries. However in this case we require that $k = O(\sqrt{\log n})$ in order to run in polynomial time. In addition, these equivalence query upper bounds are tight, in the sense that any algorithm using a polynomial number of membership queries (regardless of its computation time or hypothesis class) must use at least $k+1$ equivalence queries if k is unknown (and at least k equivalence queries if k is known).

Table 1 summarizes our specific results for k-term DNF and the other classes that we consider (these classes are defined in Section 3).

The remainder of this paper is organized as follows. In Section 2 we give additional motivation for the need to reduce the number of equivalence queries used by a learning algorithm. Then, in Section 3, we give the needed formal definitions. The technical details of the remaining sections are for the most part independent, though Section 6 parallels and refers back to Section 5. In Section 4 we present a generalization of the halving algorithm that reduces the number of equivalence queries required by making use of membership queries. This is the only positive result that we describe in which the learner may use unbounded computation time. In addition, it uses hypothesis selected from outside the given concept class. In Section 5 we give our results for learning k-term DNF formulas. In Section 6 we give our results for the class of read-k sat-j DNF formulas. In Section 7 we give our results for learning the class of monotone DNF formulas. Next, in Section 8, we present results for learning Horn sentences. In Section 9 we present a technique to reduce the number of equivalence queries needed by any concept class for which there already exists a polynomial time learning algorithm that uses membership queries and justifying assignments. A previous

Representation Class	Previous Upper Bound	New Upper Bound	Lower Bound
k-term DNF (k=$O(\log n)^*$)			
k not known	$2^{O(k)} \log n$ [BR92]	$k+1$	$k+1$ [BC92]
k known		k	k [BC92]
Monotone DNF			
m not known	$m+1$ [Ang88]	$m - \Theta(1)$	$m - \omega(1)$
m known		$m - \Theta\left(\frac{\log m + \log n}{\log n - \log\log m}\right)$	$m - \Theta\left(\frac{\log m + \log n}{\log n - \log\log m}\right)$
Read-k Sat-j DNF			
m not known	$n^{O(kj)}$ [AP92]	$m+1$	$m+1$
m known		m [‡]	m [§]
Horn Sentences			
m not known	$m(2n+1)$ [AFP90]		
m known		$O\left(\frac{mn}{\log m + \log n}\right)$ [†]	$\Omega\left(\frac{mn}{\log m + \log n}\right)$
Read-once formulas			
over $\vee, \wedge, \neg$	n [AHK89]	$O\left(\frac{n}{\log n}\right)$	$\Omega\left(\frac{n}{\log n}\right)$ [BC92]
over B_k	n [BHH92b]	$O\left(\frac{n}{\log n}\right)$	$\Omega\left(\frac{n}{\log n}\right)$ [BC92]
Arithmetic read-once formulas for $\lvert\mathcal{F}\rvert = o(n/\log n)$	n [BHH92a]	$O\left(\frac{n \log \lvert\mathcal{F}\rvert}{\log n}\right)$	$\Omega\left(\frac{n \log \lvert\mathcal{F}\rvert}{\log n}\right)$ [BC92]
DFA with n states			
n not known	n [Ang87b, RS89]	$O\left(\frac{n}{\log n}\right)$	
n known			$\Omega\left(\frac{n}{\log n}\right)$

*For $k = O(\sqrt{\log n})$ we can obtain this result using k-term DNF formulas as the hypotheses for the equivalence queries. For the remaining cases, general DNF formulas are used for the equivalence queries.

†With unlimited computation. Furthermore, we note that both this upper bound and the matching lower bound hold for arbitrary DNF formulas.

‡The hypothesis class is general DNF formulas.

§The lower bound holds for $m \leq \sqrt{j(k-1)n/2}$.

Table 1: *This table summarizes the numbers of equivalence queries in our results. All lower bounds allow the learning algorithm to use unbounded computation time and to propose any hypothesis. Unless stated otherwise, all upper bounds are for algorithms that use polynomial computation time and use hypotheses from the given class. For the Boolean classes n is the number of variables and m is the number of terms/clauses. For k-term DNF, k is the number of terms. Note that all upper bounds for an unknown size parameter can be used when the size parameter is known. Likewise, all lower bounds for a known size parameter apply when the size parameter is unknown.*

reduction [BHHK91] shows that justifying assignments for n variables can be generated with n equivalence queries (as long as the class being learned satisfies the technical condition of being projection closed). We improve that transformation, somewhat surprisingly, to show that only $n/\log n$ equivalence queries are required. This improvement decreases the number of equivalence queries required to learn read-once formulas over certain bases [AHK89, BHH92b], arithmetic read-once formulas [BHH92a] and non-monotone switch configurations [RS90], giving bounds that are asymptotically tight. Next, in Section 10 we present our results for learning deterministic finite state automaton (DFAs). Finally, we give some concluding remarks and discuss open problems.

2 Motivation

We now further describe our motivation for reducing the number of equivalence queries needed to obtain exact identification. In this work we are able to reduce (sometimes quite dramatically) the number of equivalence queries needed to obtain exact identification at the expense of increasing the computation time and number of membership queries (although they remain polynomial). Clearly being able to prove tight bounds on the number of equivalence queries needed for exact identification is of great theoretical interest. We now argue that is also of practical interest.

As one example consider the situation in which the target function f measures some observable consequence of the learner's action. For example, Rivest and Schapire [RS89] motivate the problem of learning an unknown DFA by the problem of a robot trying to learn to navigate in an environment describable by a finite state machine. Here a membership query represents experimentation by the robot, followed by an observation of its perceived state after executing the experiment. Thus, in this context, membership queries can be made in an unsupervised manner by the learner interacting with the environment. On the other hand, an equivalence query requires the intervention of a teacher to provide a counterexample. For example, there are some states that the robot can reach only through specific sequences of actions that it cannot hope to stumble on through its own experimentation, and of which it must be told by a teacher. In such settings, minimizing the number of equivalence queries allows the learner to minimize the supervision needed.

Another motivation comes from the goal of minimizing the number of prediction mistakes in an on-line learning model. As we have mentioned, the model of learning with membership and equivalence queries is essentially equivalent to the on-line learning model when the learner is provided with membership queries [Ang88, Lit88]. The conversion of an algorithm A that uses membership queries and equivalence queries to an on-line algorithm A' works as follows. If algorithm A wants to perform some internal computation or perform a membership query then algorithm A' will perform the same task. If A wants to make an equivalence query with hypothesis h, then A' can just use hypothesis h to make predictions. If hypothesis h is equivalent to the target no mistakes will occur and the learning process is done. Otherwise, if algorithm A' makes a prediction mistake on instance x, then this instance can be passed to A as a counterexample. Thus, the number of

mistakes made by A' is just one less than the total number of equivalence queries made by A. Since the primary goal of an on-line learning algorithm is to reduce the number of mistakes, the learner is willing to spend additional computation time and make additional membership queries to reduce the number of mistakes. Thus minimizing the number of equivalence queries is equivalent to minimizing prediction errors in an on-line learning model.

Another situation in which we would like to minimize equivalence queries is the case where the "learning" algorithm is being used to interpolate a function that is in fact already known in some sense (e.g. we have a black box oracle, a truth table, or a slow simulator whose performance we hope to emulate) to obtain some desired representation (e.g. a read-once formula or equivalently a circuit of fan-out 1). In this situation, membership queries may be readily implementable as substitutions, yet implementing an equivalence query may be much more expensive (or perhaps for some classes even intractable).

Clearly, there are some other situations in which it is not desirable to reduce equivalence queries at the expense of performing more membership queries. For example in the classification learning problem of fitting a function to data points, it is easy to implement an equivalence query (by testing the hypothesis on the available data), whereas implementing membership queries is often extremely difficult. Nevertheless, as we have discussed, there are many situations in which it is extremely important to minimize the number of equivalence queries needed to obtain exact identification.

3 Definitions

We now formalize the model of learning from membership and equivalence queries [Ang87a]. The learner must infer an unknown *target concept* f chosen from some known *representation class* $\mathcal{C}$, which is a set of representations of functions mapping some domain $\mathcal{X}$ into a range $\mathcal{Y}$. We typically parameterize $\mathcal{C}$ as $\mathcal{C} = \bigcup_{n \geq 1} \mathcal{C}_n$, where $\mathcal{C}_n$ is those elements of $\mathcal{C}$ that represent functions on n inputs. For almost all classes studied here $\mathcal{C}$ is some subset of Boolean formulas, n is the number of variables, $\mathcal{X} = \{0,1\}^n$, and $\mathcal{Y} = \{0,1\}$. For these classes, we assume that the n variables are $v_1, v_2, \ldots, v_n$ where the value of v_i is given by the ith bit of the instance. A *literal* is a variable v_i or its negation $\bar{v}_i$. For instance $x \in \mathcal{X}$ we use x_i to denote the i^{th} bit of x. Thus x_i gives the value for variable v_i. A We shall also refer to the instances as *assignments*, since they can be viewed as functions that assign a domain value to each variable.

A Boolean formula is said to be *monotone* if it contains no negations. A formula is in disjunctive normal form (DNF) if it is written as the disjunction of monomials (or *terms*). In addition for the classes of monotone DNF and Horn sentences we use $v_1 \cdots v_k$ to denote the term $v_1 \wedge \cdots \wedge v_k$. The principal non-Boolean class considered here is the class of DFAs. In this case n is the number of states in the target DFA, $\mathcal{X}$ consists of all strings from the given alphabet and $\mathcal{Y}$ is $\{0,1\}$.

The learning criterion expected here is that of *exact identification* which is achieved by the

learner if it can infer a concept that is logically equivalent to the target concept on all instances in $\mathcal{X}$. In addition we want the learning algorithm to be efficient. Namely, the running time of the algorithm should be bounded above by a polynomial function of the size of the smallest representation from $\mathcal{C}$ equivalent to f and of the dimension of the domain (i.e. n).[1]

Let $f(x) = 1$ denote that formula f is true for instance $x \in \mathcal{X}$ and $f(x) = 0$ denote that h is false for instance $x \in \mathcal{X}$. The learner is provided with two types of queries with which to learn about f. A *membership query* MQ(x) for $x \in \mathcal{X}$ returns "yes" if $f(x) = 1$ and returns "no" if $f(x) = 0$. An *equivalence query*, Equiv(h), takes a hypothesis $h \in \mathcal{C}$ returns "yes" if h is logically equivalent to f or returns a counterexample otherwise[2]. A *positive counterexample* x is one for which $f(x) = 1$ but $h(x) = 0$. Likewise, a negative counterexample is one for which $f(x) = 0$ but $h(x) = 1$.

If $\mathcal{C}$ is a representation class, we define $\mathcal{E}(\mathcal{C}, q)$ to be the minimum worst case number of equivalence queries made by any polynomial time algorithm that uses at most q membership queries to identify $\mathcal{C}$. (That is, an algorithm A that exactly identifies $\mathcal{C}$ and never makes more than q membership queries when doing so, must make at least $\mathcal{E}(\mathcal{C}, q)$ equivalence queries when run for some target $f \in \mathcal{C}$. Furthermore there is some such A that achieves this bound when run on any target from $\mathcal{C}$.) Observe that this quantity typically decreases as q increases. We let $\mathcal{E}(\mathcal{C})$ denote the minimum number of equivalence queries made by any polynomial time algorithm to identify $\mathcal{C}$ (making a polynomial number of membership queries). Likewise, when the learner is not restricted to use polynomial time we let $\mathcal{E}_U(\mathcal{C}, q)$ denote the minimum number of equivalence queries needed to obtain exact identification when at most q membership queries are made. Finally, $\mathcal{E}_U(\mathcal{C})$ denotes the number of equivalence queries needed to obtain exact identification using unlimited time when a polynomial number of membership queries can be made.

Here is a summary of the representation classes we study in this paper.

k-term DNF This is the class of DNF formulas having at most k terms. Angluin gave a polynomial time identification algorithm for the special case when k is constant [Ang87a], and Blum and Rudich have since given a more efficient algorithm that runs in polynomial time for $k = O(\log n)$ [BR92].

Read-k Sat-j DNF A DNF formula is a read-k sat-j DNF formula if every variable appears at most k times, and every assignment satisfies at most j terms in the formula. The class of

[1] This measure of efficiency in terms of the encoding of the target function is what makes it necessary to define learnability in terms of a class of representations rather than a class of functions. Different representation classes may represent the same class of functions with varying efficiencies. For example DFA's, which are a relatively inefficient encoding of regular sets are learnable, whereas NFA's, which represent the same set more efficiently and hence give a potential learning algorithm a smaller time budget, are not.

[2] Often the notion of an equivalence query is generalized so that the learner can propose any polynomially evaluatable hypothesis. While almost all of our positive results apply for the more stringent definition we have given, the negative results hold even under this more general model.

read-k sat-j DNF formulas was proved to be learnable by Aizenstein and Pitt [AP92]. Their algorithm uses $n^{\Theta(jk)}$ equivalence queries.

Monotone DNF This is the class of monotone DNF formulas. These were proved to be efficiently learnable by Valiant [Val84] and Angluin [Ang88]. We use m to denote the number of terms in the target formula. A polynomial time learning algorithm can use time polynomial in n and m.

Horn Sentences These are the conjunction of implications each of the form $v_{i_1} \cdots v_{i_m} \rightarrow v_j$. It is easily shown that an algorithm for learning the class of Horn Sentences can be modified to learn the class of DNF formulas in which at most one variable may appear negated per term. Thus this class can be viewed as a generalization of the class of monotone DNF formulas. Angluin, Frazier, and Pitt give a polynomial time algorithm to learn Horn sentences [AFP90]. For this class m denotes the number of Horn clauses in the sentence, and a polynomial time algorithm can use time polynomial in n and m.

Read-Once Formulas These are Boolean formulas in which each variable may appear just once (i.e. at a single leaf when the formula is expressed as a tree with operations on the internal nodes and variables at the leaves). Angluin, Hellerstein, and Karpinski give a polynomial time exact identification algorithm for the formulas over the operators (or *basis*) $\{\vee, \wedge, \neg\}$ [AHK89], and Bshouty, Hancock, and Hellerstein have extended this to more complicated sets of Boolean functions, including the set B_k of all functions on k or fewer inputs (for an arbitrary constant k) [BHH92b]. Our results also apply for the class of non-monotone switch configurations [RS90] and arithmetic read-once formulas over the basis of addition, subtraction, multiplication and division over a field $\mathcal{F}$ [BHH92a].

DFAs These are deterministic finite automata representing regular languages over some alphabet Σ. These languages can be viewed as functions from Σ^* to $\{0, 1\}$, and an efficient exact identification algorithm is due to Angluin [Ang87b] and has since been improved by Rivest and Schapire [RS89]. Here we let n denote the number of states in the target automaton.

A variation that we explore here is whether the learner is given the size of the target representation before the learning session begins (i.e. for the Boolean classes is m known, and for DFAs is n known?). For previous work aimed mainly at proving tractability, this is not an important distinction, since a generic transformation generic transformation conversion from an algorithm that knows the size of the target to one that does not [HKLW88]. However for our precise bounds this difference can be important, and for some classes we obtain different results depending on whether or not the size of the target is known a priori. This issue does not affect the asymptotic results for DFA's, nor does it matter for read-once formulas (whose size is always $O(n)$). But for monotone DNF it becomes significant in the case where m may be super-polynomial in n, and for

k-term DNF it turns out that an extra query is necessary and sufficient if k is unknown. For Horn sentences the relationship is more complex.

In this paper log denotes the logarithm base 2, and ln denotes the natural logarithm.

4 A Generalization of the Halving Algorithm

In this section we consider a generalization of the halving algorithm [BF72, Lit88] in which we can reduce the number of equivalence queries required by allowing the learner to make membership queries. In the section we do not bound the computation time of the learner, and the hypotheses proposed need not come from $\mathcal{C}$. However, the learner is still limited to make a polynomial number of membership queries.

Theorem 1 *For any concept class $\mathcal{C}$ and any $q \geq 2 \ln |\mathcal{C}|$, $\mathcal{E}_U(\mathcal{C}, q) \leq \frac{\log |\mathcal{C}|}{\log q - \log \ln |\mathcal{C}|}$.*

Proof: For a set of concepts $\mathcal{C}$ and $\xi \in \{0,1\}$ we define $\mathcal{C}^{\xi}(x_0) = \{f \in \mathcal{C} \mid f(x_0) = \xi\}$. In Figure 1 we give an algorithm to learn any class $\mathcal{C}$ with unlimited computation time using at most $\log |\mathcal{C}| / \log \frac{1}{1-\alpha}$ membership queries and $\log |\mathcal{C}| / \log \frac{1}{\alpha}$ equivalence queries for any $0 < \alpha \leq 1/2$. In this algorithm, a membership query is performed if there exists an instance $x \in \mathcal{X}$ for which both $\mathcal{C}_i^0(x)$ and $\mathcal{C}_i^1(x)$ have cardinality at least $\alpha|\mathcal{C}_i|$. Thus, each membership query allows the learner to eliminate at least $\alpha|\mathcal{C}_i|$ of the remaining concepts. If for all $x \in \mathcal{X}$ either $\mathcal{C}_i^0$ or $\mathcal{C}_i^1$ has cardinality less than $\alpha|\mathcal{C}_i|$ then, just like in the standard halving algorithm, the learner uses the majority vote hypothesis. However, instead of just being assured that half of the elements of $\mathcal{C}_i$ are eliminated, here, at least $(1-\alpha)|\mathcal{C}_i|$ concepts are eliminated. Thus it immediately follows that this algorithm uses at most $q = \log |\mathcal{C}| / \log(1/(1-\alpha))$ membership queries and at most $\log |\mathcal{C}| / \log(1/\alpha)$ equivalence queries.

Using the standard inequality $x \log e \leq \log\left(\frac{1}{1-x}\right)$ it follows that $q \leq \frac{\log |\mathcal{C}|}{\alpha \log e}$, which we rewrite as $\alpha \leq \ln |\mathcal{C}|/q$. Substituting this upper bound on α into the upper bound on equivalence queries derived above gives the bound claimed for $\mathcal{E}_U(\mathcal{C}, q)$. □

For most of the classes we study, $|\mathcal{C}|$ is exponential, thus by setting $q = \frac{\log^{d+1} |\mathcal{C}|}{\log e}$ (for $d \geq 1$) it immediately follows that in such cases, we can reduce the number of equivalence queries to $\frac{\log |\mathcal{C}|}{d \log \log |\mathcal{C}|}$ from the $\log |\mathcal{C}|$ of the halving algorithm, while still using only a polynomial number of membership queries.

Observe that for Horn Sentences $|\mathcal{C}| = O((n+1)^m 2^{mn})$, and for arbitrary DNF formulas $|\mathcal{C}| = O(3^{mn})$. Furthermore, for both classes it can be shown that $|\mathcal{C}| = \Omega(2^{cmn})$ for some constant c. To see this lower bound consider the class of monotone DNF formulas in which each term contains exactly $n/2$ variables. For this class there are $\binom{n}{n/2}$ possible terms, and all possible subsets of these m terms represent logically distinct formulas. Thus we obtain the following corollary.

Gen-Halving$(\mathcal{C}, \alpha)$

Initialize $\mathcal{C}_0$ to $\mathcal{C}$ and i to 0.
While $|\mathcal{C}_i| > 1$
    If there exists $x \in \mathcal{X}$ such that $\min\left\{|\mathcal{C}_i^0(x)|, |\mathcal{C}_i^1(x)|\right\} \geq \alpha|\mathcal{C}_i|$
        $\xi \leftarrow \mathrm{MQ}(x)$
        $\mathcal{C}_{i+1} \leftarrow \mathcal{C}_i^{\xi}(x)$
    Else
        For all $x \in \mathcal{X}$
            If $|\mathcal{C}_i^1(x)| \geq |\mathcal{C}_i^0(x)|$ then $g(x) \leftarrow 1$
            Else $g(x) \leftarrow 0$
        If Equiv(g) = "yes" then return g
        Else Let $x \leftarrow$ Equiv(g)
            $\mathcal{C}_{i+1} \leftarrow \mathcal{C}_i^{1-g(x)}(x)$
Return $\mathcal{C}_i$

Figure 1: An generalization of the halving algorithm that uses membership queries to reduce the number of equivalence queries.

Corollary 2 *For $\mathcal{C}$ the class of Horn sentences or the class of DNF formulas,*

$$\mathcal{E}_U(\mathcal{C}) = O\left(\frac{mn}{\log n + \log m}\right).$$

In Section 16 we show that both of these results are asymptotically tight, by giving a matching lower bound for the class of Horn Sentences.

5 k-term DNF Formulas

Bshouty and Cleve [BC92] prove that k equivalence queries are required to learn a k-term DNF formula when the learner knows k a priori, and that $k+1$ queries are required when k is not known in advance (the extra query comes because the algorithm does not know when to stop looking for new terms). We now give algorithms that match these lower bounds.

In this section, we first present an algorithm for learning k-term DNF formulas in which the running time and the number of membership queries are $O\left(n(\log n)^{O(1)}2^{O(k)}\right)$, which is polynomial when $k = O(\log n)$. In this algorithm the hypotheses for the equivalence queries are general DNF formulas. The number of equivalence queries when k is known is k, and $k+1$ otherwise. We then present an algorithm that uses the same number of equivalence queries as above, but for which the hypotheses are k-term DNF formulas. For this algorithm the running time and the number of membership queries are $n2^{O(k^2)}(\log n)^{O(k)}$, which is polynomial when $k = O(\sqrt{\log n})$.

Blum-Rudich Alg

Let x be a positive example of f
Initialize $h \leftarrow$ `false`.
Repeat
    Call **Produce-terms**(x).
    Add the terms generated by **Produce-terms** to h.
    Repeat
        Perform Equiv(h)
            If answer "yes" then return h
            Else let y be the counterexample
                If $f(y) = 0$, remove from h all terms t for which $t(y) = 1$
    Until $f(y) = 1$
    $x \leftarrow y$
Until done

Figure 2: The deterministic version of the Blum-Rudich algorithm to learn k-term DNF formulas.

Our algorithms are based on the algorithm of Blum and Rudich [BR92], which we now briefly summarize. The key processing of the Blum-Rudich algorithm can be encapsulated as a procedure **Produce-terms** that when given a positive example x, produces $c = 2^{O(k)}(\log n)^{O(1)}$ terms, one of which is in the target formula f[3]. Furthermore, this term is satisfied by x. This procedure runs in time nc and uses at most nc membership queries, and no equivalence queries. The deterministic version of their algorithm works as shown in Figure 2 where $f = T_1 \vee T_2 \vee \cdots \vee T_k$. Their algorithm uses $2^{O(k)}(\log n)^{O(1)}$ equivalence queries, mainly to produce the needed negative counterexamples. Our goal is to reduce this number to $k + 1$.

Observe that most of the equivalence queries used by the Blum and Rudich algorithm are used to produce the needed negative counterexamples. We reduce this number to $k + 1$ by simulating the negative counterexamples. Suppose we call **Produce-terms**(x), for a positive counterexample x, and let $\mathcal{T} = \{T_1, \ldots, T_c\}$ be the terms returned. Our goal is to drop from $\mathcal{T}$ any term T that does not imply f (i.e. is satisfied by some negative example for f). If we can achieve this, and hence add to h only those terms that imply f, then when we next ask Equiv(h) we are guaranteed to get a positive counterexample. Let T be a term (not equivalent to `false`) and let h be a DNF formula. We define the projection h_T of h as the DNF formula obtained by replacing every variable $v \in T$ by 0 if it is negated and by 1 otherwise. We have the following lemmas:

Lemma 1 *Let h be a DNF formula and T_0 a term. Then T_0 implies h iff h_{T_0} is a tautology (i.e. equivalent to* `true`*).*

[3]Technically, the given statement just holds where f is some k-term DNF formula that is logically equivalent to the target.

Our $O(\log n)$-term DNF Algorithm

Let x be a positive example of f
Initialize $h \leftarrow$ `false`.
Repeat
    Call **Produce-terms**(x), and let $\mathcal{T} = \{T_1, \ldots, T_c\}$ the terms returned.
    Add to h those terms in $\mathcal{T}$ that imply f.
    Perform Equiv(h)
        If the answer is "yes" then we are done.
        Else let x be the (positive) counterexample returned.
Until done.

Figure 3: Our refinement to Blum-Rudich Algorithm.

An $(n,k)-$universal set is a set of Boolean n-tuples $\{b_1, \ldots, b_t\} \subseteq \{0,1\}^n$ such that every subset of k bit positions (or variables, in our context) assumes all of its 2^k possible assignments in the b_i's. Naor and Naor [NN90] give an explicit construction of an $(n,k)-$universal set of size $t = O(k2^{3k}\log n)$.

Lemma 2 *Let S be an (n,k)-universal set, and let f be a k-term DNF formula. Then f is a tautology if and only if $f(a) = 1$ for all $a \in S$.*

The proofs of these lemmas are trivial and thus are omitted here. Thus, to remove the terms in $\mathcal{T}$ that do not imply f, we simply use the (n,k)-universal set to check if a term T implies f. Our algorithm is described in more detail in Figure 3. Observe that the number of membership queries needed to remove such terms is just $O(k2^{3k}\log n)$ — the size of the universal set.

Since each iteration of our algorithm h implies f, after k calls to **Produce-terms** h will contain all terms of f, therefore f implies h, so $h \equiv f$. If k is known then there is no need for the $(k+1)$st equivalence query. So the number of equivalence queries is $k+1$ if k is not known, and k if it is known. The number of membership queries is $O\left(n(\log n)^{O(1)}2^{O(k)}\right)$. Thus we have proved the following theorem.

Theorem 3 *There is a polynomial time algorithm that exactly identifies an unknown k-term DNF formula using $O\left(n(\log n)^{O(1)}2^{O(k)}\right)$ membership queries and k equivalence queries whose hypotheses may be arbitrary DNF formulas (or $k+1$ queries, if k is not given as an input to the algorithm).*

Now we present an alternate algorithm that in addition to reducing the number of equivalence queries to $k+1$ (or k, if k is known a priori), also makes these queries on hpyotheses that are k-term DNF formulas (versus arbitrary DNF formulas). We first describe a parallel version of our

algorithm in which there are $k+1$ parallel rounds of equivalence queries, but the total number of queries is $c^{O(k)}$, where $c = 2^{O(k)}(\log n)^{O(1)}$. We then show how to make the algorithm sequential in such a way to reduce the number of equivalence queries to $k+1$ (by reducing the number of equivalence queries in each round to 1).

5.1 A Parallel Greedy Algorithm

We begin with an informal description of our parallel algorithm. Let x be a positive example of f. If we call **Produce-terms**(x) we get c terms $\mathcal{T}^{(1)} = \{T_1^{(1)}, \ldots, T_c^{(1)}\}$, one of which is guaranteed to be a term in f (without loss of generality say T_1). We now continue performing the following step in parallel on all these terms. For each $T \in \mathcal{T}^{(1)}$ make the equivalence query Equiv(T). If the counterexample x is negative then T is a "bad" term and thus we can quit working on it. Otherwise (so $f(x) = 1$), call **Produce-terms**(x) to get another c terms, $\mathcal{T}^{(2)} = \{T_1^{(2)}, \ldots, T_c^{(2)}\}$, one of which is guaranteed to be a term in f. Furthermore, since $T_1 \in \mathcal{T}^{(1)}$, it follows that some *other* term from f (say T_2) is in $\mathcal{T}^{(2)}$.

We now work in parallel on all formulas of the form $T_i^{(1)} \vee T_j^{(2)}$ for $1 \leq i, j \leq c$ making an equivalence query for each one. As before, if the counterexample is negative we stop working on that formula. Otherwise we give the counterexample as input to **Produce-terms** and get another c terms one of which is a *new* term in f. After k such parallel phases we will have a set of k-term DNF formulas, one of which is the target formula. Finally, we use equivalence queries to find which formula is the target. Thus in summary, there are k phases and in phase i there are at most c^i i-term DNF formulas, one of which contains i terms from f. In addition, note that we get these formulas independently, in the sense that getting some i-term DNF formula does not depend on getting other i-term DNF formulas.

We now analyze the complexity of this parallel algorithm. The total number of DNF formulas produced in the k phases (and thus the total number of equivalence queries made) is at most $\sum_{i=1}^{k} c^i \leq c^{O(k)}$. The number of membership queries is $nc^{O(k)}2^{O(k)}(\log n)^{O(1)}$. Thus the above algorithm learns k-term DNF formulas in Angluin's restricted model (where each hypothesis comes from the class being learned, k-term DNF) in sequential time $nc^{O(k)}2^{O(k)}(\log n)^{O(1)} = n2^{O(k^2)}(\log n)^{O(k)}$.

5.2 Reducing the Number of Equivalence Queries

The idea for reducing the number of equivalence queries is the following. Suppose we have two i-term DNF formulas h and h', and we want to run an equivalence query for both. Instead, we test whether $h \equiv h'$. If this is the case then we can drop one of them. Otherwise, we find an assignment y for which (without loss of generality) $h(y) = 0$ and $h'(y) = 1$. We then perform a membership query to see if y is a negative or positive example. If y is a negative example then h' can be discarded and we ask an equivalence query with h. Otherwise y is a positive counterexample for h and we perform an equivalence query for h'.

Using this idea we reduce the number of equivalence queries in phase i from c^i to 1 (the last i-term DNF formula has no other i-term DNF formula to be compared with, so we ask an equivalence query with it). On the other hand, the number of membership queries is increased by $c^i - 1$. If k is known then there is no need to ask an equivalence query in the kth phase, because the formula to pass the last test is guaranteed to be the target formula. Otherwise, we need an equivalence query for the kth phase as well, and then the number of equivalence queries is $k+1$.

All that remains now is to give an algorithm that tests whether two k-term DNF formulas are equivalent. We shall use the following lemma in addition to Lemmas 1 and 2.

Lemma 3 *Let $h = T_1 \vee \cdots \vee T_j$ and $h' = T'_1 \vee \cdots \vee T'_{j'}$ be DNF formulas. Then $h \equiv h'$ iff $T_i \Rightarrow h'$ and $T'_{i'} \Rightarrow h$, for each $1 \leq i \leq j$ and $1 \leq i' \leq j'$.*

The proof of this lemma is trivial and thus is omitted here. By Lemma 1 we reduce the problem of testing these implications (or finding a counterexample) to that of testing tautology for a k-term DNF formula. As above in Lemma 2, this is efficiently done by checking the elements of an (n,k)-universal set.

Theorem 4 *There is an algorithm that exactly identifies an unknown k-term DNF formula using $n2^{O(k^2)}(\log n)^{O(k)}$ membership queries, $n2^{O(k^2)}(\log n)^{O(k)}$ time, and k equivalence queries (or $k+1$, if k is not given as an input to the algorithm).*

6 Read-k Sat-j DNF formulas

A DNF formula is *read-k* if every variable appears in it at most k times, and it is *sat-j* if every assignment satisfies at most j terms in it. The class of read-k sat-j DNF formulas was proven to be learnable by Aizenstein and Pitt [AP92]. The running time of their algorithm is $O(n^{4kj+2j+2})$ and the algorithm uses at most $k(n^{j+2} + n^{j+1})$ membership queries and at most $kn^{2kj+j+1}$ equivalence queries.

In Section 6.1 we show how to modify the Aizenstein and Pitt algorithm to decrease the number of equivalence queries to m (the number of terms in the target formula) when m is given a priori to the learner, or to $m+1$ if m is not known. Both the running time and the number of membership queries in our algorithm are $n^{\Theta(kj)}$. In Section 6.2 we show that the number of terms in a read-k sat-j DNF formula is bounded above by $4\sqrt{jk(k-1)n}$, for $k > 1$, and by j for $k = 1$. In Section 6.3 we present a lower bound on the number of equivalence queries needed to learn read-k sat-j DNF formulas. The lower bound is m (or $m+1$, if m is not a priori) for $m \leq \sqrt{j(k-1)n/2}$ and $k > 1$, and it is j (or $j+1$, if j is unknown a priori) for $k = 1$. The first lower bound holds for $jk = o(n/(\log n)^2)$, and the second holds for $j = o(n/\log n)$.

Let Read-k Sat-j DNF$_n$ represent the class of read-k sat-j DNF formulas on n variables, and let Read-k Sat-j DNF$_{n,m}$ be the subclass of those formulas that have at most m terms.

6.1 An Upper Bound for Read-k Sat-j DNF Formulas

The algorithm of Aizenstein and Pitt [AP92] shares the same high level structure as the Blum and Rudich algorithm to learn k-term DNF formulas, shown in Figure 2. Recall that the main processing of that algorithm is encapsulated in a subroutine **Produce-terms** that takes a positive example x and produces a set of terms, one of which is in the target formula f and is satisfied by x. For this version of **Produce-terms**, the set of terms returned is of size $n^{O(kj)}$, and the dominant use of equivalence queries is to provide negative examples that eliminate incorrect terms. As with the k-term DNF algorithm, we eliminate the need for such equivalence queries by showing that polynomial time computation and membership queries can detect whether a term indeed implies the target formula (in the sense that only positive examples can satisfy the term). Since each call to **Produce-terms** returns a set of terms that includes one from the target formula not present in the previous hypothesis, it follows that the algorithm needs only m positive counterexamples. Thus by removing the need for negative examples we prove our bound (if m is not known a priori the algorithm needs an additional final query to detect equivalence before termination).

We first present some definitions from Aizenstein and Pitt. A term is *almost satisfied by an assignment x with respect to a literal v* if v is the only literal in T that is assigned 0 by x. We denote the assignment x with literal v fixed to 0 (respectively, 1) as $x_{v \leftarrow 0}$ (respectively, $x_{v \leftarrow 1}$). The *sensitive set of x* is defined by $\mathit{sensitive}(x) = \{\text{literal } v \mid x \text{ assigns 1 to } v \text{ and } f(x) \neq f(x_{v \leftarrow 0})\}$. Thus, if $v \in \mathit{sensitive}(x)$, flipping v in x changes the value of f. An instance x is said to be an *i-variant* of an instance y if the number of bits on which x and y disagree is at most i. For a term T, let $\mathit{lits}(T)$ denote the set of literals in T.

Thus we need just show how to determine whether a term T logically implies an unknown read-k sat-j DNF formula f (denoted $T \Rightarrow f$). Given a term T and a DNF formula f, we define the projection f_T to be the formula obtained from f by replacing every literal that appears in T by the constant 1 and every literal whose complement appears by the constant 0.

Let T be a term and let f be the target read-k sat-j DNF formula. By Lemma 1, $T \Rightarrow f$ if and only if f_T is a tautology. We now show a series of lemmas that lead up to a result showing that we can efficient test whether f_T is a tautology using membership queries (without knowing f).

Lemma 4 *Let f be a read-k sat-j DNF formula and let T be a term. Then f_T is a read-k sat-j DNF formula.*

Proof: Since f_T is formed from f by deleting terms and literals, clearly it is read-k. For an example x, let x_T be the example obtained by setting all literals that appear in T so that they are satisfied We prove that f_T is read-j by observing that if x satisfies $j' > j$ terms in f_T, then x_T satisfies j' corresponding terms in f, which is a contradiction. □

Thus, equivalence testing is reduced to testing via membership queries whether a read-k sat-j DNF formula is a tautology. Before showing how this is done, we first give two preliminary results

(the first restates Lemma 7 of Aizenstein and Pitt).

Lemma 5 ([AP92]) *Let x be an example satisfying a read-k sat-j DNF formula f. There are at most $2kj$ literals v for which there is a term in f almost satisfied by x with respect to $\bar{v}$.*

The following analogous lemma covers the case when x falsifies f.

Lemma 6 *Let x be any assignment falsifying a read-k sat-j DNF formula f. There are at most $2(k+1)j$ literals v for which there is a term in f almost satisfied by x with respect to $\bar{v}$.*

Proof: Let T be the term that is satisfied by example x and no other instances (i.e. T contains literal v_i if $x_i = 1$ and literal $\bar{v}_i$ if $x_i = 0$). Now let $f' = f \vee T$. Note that f' is read-$(k+1)$ since f is read-k and by adding T we added one occurrence for every variable. Also note that f' is sat-j (since none of the examples that satisfy any terms in f satisfy T, and vice versa). Now x is a positive example for f', so Lemma 5 states that there exist at most $2(k+1)j$ literals v for which there is a term in f' that is almost satisfied by x with respect to $\bar{v}$. Since every term in f is also in f', the lemma follows. □

We now state the result that allows efficient tautology testing on read-k sat-j formulas.

Theorem 5 *Let f be a read-k sat-j DNF formula and let x be* any *assignment. Then f is a tautology iff $f(x') = 1$ for every $2(k+1)j$-variant x' of x.*

Proof: We consider the two directions of the claim.

($\Rightarrow$) This direction is trivial, for if f is a tautology then its value is 1 on every input.
($\Leftarrow$) Suppose f in not a tautology. We will produce a $2(k+1)j$-variant of x that falsifies f.

Since f is not a tautology, there is an example y for which $f(y) = 0$. If y is a $2(k+1)j$-variant of x then we are done. So assume that y is not a $2(k+1)j$-variant of x. Let

$$\begin{aligned} V(y) &= \{\text{literal } v \mid \text{there is a term in } f \text{ that is almost satisfied by } y \text{ with respect to } \bar{v}\}, \\ D(x,y) &= \{\text{literal } v \mid x \text{ assigns 0 to } v \text{ and } y \text{ assigns 1 to it}\}. \end{aligned}$$

By Lemma 6 we have that $|V(y)| \leq 2(k+1)j$, and by our assumption that y is not a $2(k+1)j$-variant of x, $|D(x,y)| > 2(k+1)j$. Therefore, there exists a literal $v \in D(x,y) - V(y)$. Since y falsifies all terms in f and since $v \notin V(y)$, the assignment $y' = y_{v \leftarrow 0}$ still falsifies f. Moreover, by Lemma 6, we know that $|V(y')| \leq 2(k+1)j$, and $|D(x,y')| = |D(x,y)| - 1$, since y' and x both assign 0 to v. If $|D(x,y')| > 2(k+1)j$, we can repeat the same process: find a literal in $v \in D(x,y') - V(y')$, flip it in y' to get a new assignment y'' that falsifies f, and so on. This process can be repeated until we get an assignment x' that falsifies f and for which $|D(x,x')| \leq 2(k+1)j$. This x' is a $2(k+1)j$-variant of x (since $|D(x,x')| \leq 2(k+1)j$), and it falsifies f, so we are done. □

To summarize our algorithm to learn read-k sat-j DNF, we modify the processing of the main loop to consider only those terms returned by **Produce-terms** that logically imply the target

formula. In order to test if a term T implies f, we check if $f_T(x') = 1$ for every $2(k+1)j$-variant x' of *any* assignment x (e.g. pick x to be all 0's). The number of membership queries needed for every test is $\binom{n}{2(k+1)j} \leq n^{2(k+1)j}$. **Produce-terms** is called $m \leq kn$ times (m is the number of terms in f), and every time it returns at most $\binom{n}{2kj}$ terms. Therefore, the number of additional membership queries needed is at most $n^{2(k+1)j} n^{2kj} = n^{4kj+2j}$. The additional running time is clearly $n^{\Theta(kj)}$.

Thus we have proved the following theorem.

Theorem 6 *$\mathcal{E}$(Read-k Sat-j DNF$_n$) $= m+1$ and $\mathcal{E}$(Read-k Sat-j DNF$_{n,m}$) $= m$. for j, k constant.*

This result is obtained using arbitrary DNF hypothesis. A trick analogous to that of the previous section can be applied to get an algorithm that uses only read-k sat-j hypothesis, but the algorithm will no longer be polynomial time in this case (since both the size of the set returned by **Produce-terms** and the number of times that routine is called can grow at least linearly in n).

6.2 Number of Terms in a Read-k Sat-j DNF Formula

A *disjoiner between* two terms T and T' is a literal that appears positively in one of them but negatively in the other. If a literal ℓ is a disjoiner between T and T', then we say that l *disjoins* T and T'. Obviously, if there is a disjoiner between two terms, there cannot be an assignment that satisfies both. Conversely, if there is no disjoiner between any two terms of a set S of terms, then there is an assignment that satisfies all the terms in the set S.

Clearly in a read-Once sat-j DNF formula f there cannot be any disjoiners, since otherwise some variable must appear twice. Therefore, there are at most j terms in f, since otherwise there would be an assignment that satisfies all of them contradicting the fact that f is Sat-j. We thus have proved the following theorem.

Theorem 7 *The number of terms in a read-once sat-j DNF formula is at most j.*

We now prove that for $k > 1$, the number of terms in a read-kn sat-j DNF formula is at most $4\sqrt{jk(k-1)n}$.

Let $G_f = (V, E)$ be the graph, induced by f, defined as follows. V is the set of terms in f, and $(u, v) \in E$, $u \neq v$, if and only if the terms u and v share some variable. An edge $(u, v) \in E$ is labeled with the set of variables that is shared by u and v.

Lemma 7 *G_f has the property that* any *$j+1$ distinct vertices in G_f contain a pair of vertices that are connected by an edge.*

Proof: Suppose the claim is not true. This means that there are $j+1$ terms in f for which no two share a variable. Therefore we can define an assignment that satisfies all of these $j+1$ terms, contradicting the fact that f is Sat-j. □

We are interested in bounding from below the number of edges in G_f (i.e. $|E|$). For this purpose we look at the complement graph of G_f, $\overline{G}_f = (V, \overline{E})$. Let K_i denote the clique on i vertices.

Lemma 8 *$\overline{G}_f$ does not contain K_{j+1}.*

Proof: Suppose this is not true. That is, there is a set of vertices $v_{i_1}, \ldots, v_{i_{j+1}}$ that perform K_{j+1}. By definition of $\overline{G}_f$, there is no edge in G_f between any two vertices among $v_{i_1}, \ldots, v_{i_{j+1}}$, contradicting Lemma 7. □

We now upper bound $\left|\overline{E}\right|$, thus giving us a lower bound on $|E|$. A graph is *i-partite* if its vertices can be partitioned into i subsets so that no edge has both ends in any one subset (we refer to the subsets as *partitions*). A graph is *complete i-partite* if it is simple, i-partite and if every vertex in any partition is connected to all vertices outside the partition. Let $T_{i,p}$ denote the complete i-partite graph on p vertices in which each partition has either $\lfloor p/i \rfloor$ or $\lceil p/i \rceil$ vertices. For a graph G, we use $\epsilon(G)$ to denote the number of edges in G.

We use the following standard lemmas. (For example see Theorem 7.9 and Exercise 1.2.9 from Bondy and Murty [BM76].

Lemma 9 *If a graph G is simple and contains no K_{j+1}, then $\epsilon(G) \leq \epsilon(T_{j,m})$, where m is the number of vertices in G.*

Lemma 10 *$\epsilon(T_{j,m}) = \binom{m-h}{2} + (j-1)\binom{h+1}{2}$ where $h = \lfloor m/j \rfloor$.*

We are ready now to find an upper bound on the number of edges in $\overline{G}_f$.

Lemma 11

$$|\overline{E}| \leq \frac{m(m+2)(j-1)}{2j}.$$

Proof: By Lemma 8, $\overline{G}_f$ does not contain K_{j+1}, and thus by Lemmas 9 and 10 it follows that:

$$\begin{aligned}
|\overline{E}| &\leq \binom{m - \lfloor m/j \rfloor}{2} + (j-1)\binom{\lfloor m/j \rfloor + 1}{2} \\
&= \frac{(m - \lfloor m/j \rfloor)(m - \lfloor m/j \rfloor - 1)}{2} + (j-1)\frac{(\lfloor m/j \rfloor + 1)(\lfloor m/j \rfloor)}{2} \\
&\leq \frac{(m - (m/j) + 1)(m - (m/j))}{2} + (j-1)\frac{((m/j)+1)(m/j)}{2} \\
&= \frac{m(m+2)(j-1)}{2j}.
\end{aligned}$$

□

We now lower bound $|E|$.

Lemma 12

$$|E| \geq \frac{m(m-3j+2)}{2j}.$$

Proof: Since that the number of vertices in G_f is m, and G_f is the complement graph of $\overline{G}_f$, $|E| + |\overline{E}| = \binom{m}{2}$. Thus by applying Lemma 11, we obtain

$$\begin{aligned} |E| &\geq \binom{m}{2} - \frac{m(m+2)(j-1)}{2j} \\ &= \frac{m(m-3j+2)}{2j}. \end{aligned}$$

□

We now upper bound on $|E|$, and then by applying the previous lemma, we can obtain an upper bound on m.

Lemma 13 $|E| \leq \binom{k}{2}n$.

Proof: Assume for contradiction that $|E|$ (and thus the number of labels in the graph is greater than $\binom{k}{2}n$. Since there are n variables, by the pigeonhole principle we conclude that there is a variable v that appears in more than $\binom{k}{2}$ labels. Let $e_1, \ldots, e_i$ be the edges whose labels contain v (so $i > \binom{k}{2}$). Let $G'(V', E')$ the subgraph of G_f whose set of edges E' is $e_1, \ldots, e_i$, and a vertex u is in V' if an only if u is an end of some edge in E'. Since any simple graph with k vertices, contains at most $\binom{k}{2}$ edges, it follows that the number of vertices in G' is greater than k. Finally, since every vertex in V' is adjacent to an edge in E' and variable v appears in the labels of all edges in E', it follows that the number of vertices in V' is greater than k, so v appears in more than k terms contradicting the fact that f is read-k. □

Theorem 8 *Let m be the number of terms in a read-k sat-j DNF formula f ($k > 1$), then $m \leq 4\sqrt{jk(k-1)n}$.*

Proof: By Lemma 12, $|E| \geq \frac{m(m-3j+2)}{2j} > \frac{m(m-3j)}{2j}$. Combining this inequality with that from Lemma 13 and solving for m yields:

$$\begin{aligned} m &\leq \sqrt{jk(k-1)n + \frac{9}{4}j^2} + \frac{3}{2}j \\ &\leq \sqrt{jk(k-1)n + \frac{9}{4}jkn} + \frac{3}{2}\sqrt{jkn} \\ &\leq \sqrt{jk(k-1)n + \frac{9}{4}jk(k-1)n} + \frac{3}{2}\sqrt{jk(k-1)n} \\ &\leq 4\sqrt{jk(k-1)n} \end{aligned}$$

where we use the fact that since a read-k formula over n variables has at most kn literals, $j \leq kn$, and the observation that for $k \geq 2$, $jkn \leq jk(k-1)n$.

6.3 Lower Bound on the Number of Equivalence Queries

We separate between two cases. The first case read-k sat-j DNF, when $k > 1$, and the second case is read-once Sat-j DNF.

6.3.1 The Case $k > 1$

We remind the reader that the variables over which the formulas are defined are $v_1, \ldots, v_n$. We use v_i^0 to denote v_i, and we use v_i^1 to denote $\bar{v}_i$. Thus $v_i^{c_i}$ is v_i, if $c_i = 0$ and is $\bar{v}_i$ if $c_i = 1$. If a variable is negated then we say its *sign* is 1, otherwise its sign is 0. Also, recall that a disjoiner between two terms is a variable that appears positively in one of them but negatively in the other.

We define a subclass $\mathcal{C}'$ of read-k sat-j DNF formulas ($k > 1$) and prove that the learner must ask at least $\sqrt{j(k-1)n/2}$ equivalence queries to learn $\mathcal{C}'$.

The Definition of the Target Class

Let $s = \left\lfloor \sqrt{\frac{n}{2j(k-1)}} \right\rfloor$, and let $m' = 1 + (k-1)s$. A formula $f \in \mathcal{C}'$ is of the form $f = f_1 \vee f_2 \vee \cdots \vee f_j$, where each *subformula* f_i is a read-k sat-once DNF formula. Each variable must appear in only one subformula, and each subformula has m' terms. To ensure f_i is sat-once, every term in f_i must contain a literal that disjoins it from every other term in f_i. We first describe f_1. The other subformulas are defined similarly over *different* sets of variables.

Every term T in f_1 has two parts: the *necessary* part and the *new* part. The necessary part contains only the disjoiners. The new part contains s literals none of which appears in the new part of any other term. Let $T_1, \ldots, T_{m'}$ be the terms in f_1. We use Π_i to denote the new part of term T_i. Namely,

$$\Pi_i = \bigwedge_{j=(i-1)s+1}^{is} v_j^{c_j},$$

for some $c_{(i-1)s+j} \in \{0,1\}$, $j = 1, \ldots, s$. Besides Π_i, T_i contains literals to disjoin it from $T_1, \ldots, T_{i-1}$. The first literal in Π_i (i.e. $v_{(i-1)s+1}^{c_{(i-1)s+1}}$) is used to disjoin T_i from the next consecutive $k-1$ terms. Then we cannot use this variable anymore. In order to disjoin T_i from the following $k-1$ terms we use the second literal in Π_i, that is $v_{(i-1)s+2}^{c_{(i-1)s+2}}$, and then we use the third literal, etc. For ease of notation we let Π^p denote the negation of the pth literal in Π.

To summarize, the terms in f_1 are:

$$\begin{aligned}
T_1 &= \Pi_1 \\
T_2 &= \Pi_1^1 \wedge \Pi_2 \\
&\cdots \\
T_{(k-1)} &= \Pi_1^1 \wedge \Pi_2^1 \wedge \Pi_{k-2}^1 \wedge \Pi_{k-1} \\
T_k &= \Pi_1^2 \wedge \Pi_2^1 \wedge \Pi_{k-2}^1 \wedge \Pi_{k-1}^1 \wedge \Pi_k \\
T_{k+1} &= \Pi_1^2 \wedge \Pi_2^2 \wedge \Pi_{k-2}^1 \wedge \Pi_{k-1}^1 \wedge \Pi_k^1 \wedge \Pi_{k+1} \\
&\cdots \\
T_{m'} &= \Pi_1^s \wedge \Pi_2^s \wedge \cdots \wedge \Pi_{m'-1}^1 \wedge \Pi_{m'}.
\end{aligned}$$

Note that every literal in T_1 appears in T_1 and in $k-1$ other terms. So the number of terms, $m' = 1 + (k-1)s$. Note also that there is no assignment that can satisfy any two of the above terms, because each two are disjoined by a disjoiner. Therefore, f_1 is indeed read-k sat-once. Finally, observe that the number of variables appearing in f_1 is sm'.

The other subformulas $(f_2, \ldots, f_j)$ are defined like f_1 but with a distinct set of variables. More specifically, f_1 is defined over the variables $v_1, \ldots, v_{sm'}$, f_2 is defined over the next consecutive sm' variables, that is $v_{sm'+1}, \ldots, v_{2sm}$, etc.

Observe that since each subformula is Sat-Once, $f = f_1 \vee \cdots \vee f_j$ is Sat-j. Also, since each subformula is defined over a distinct set of variables, and since every one is read-k, it is the case that f is read-k. Finally, the number of variables used in f is

$$jsm' = js + j(k-1)s^2 \leq j\sqrt{\frac{n}{2j(k-1)}} + j(k-1)\left(\sqrt{\frac{n}{2j(k-1)}}\right)^2 = \sqrt{\frac{n}{2(k-1)}} + \frac{n}{2}.$$

For $k \geq 2$ it is true that $\sqrt{\frac{n}{2(k-1)}} \leq \frac{n}{2(k-1)} \leq \frac{n}{2}$, therefore the number of variables used in f is less than n, so f is well-defined.

Observe also that the number of terms in f is

$$m = jm' = j(1 + (k-1)s) = j\left(1 + (k-1)\sqrt{\frac{n}{2j(k-1)}}\right) > j(k-1)\sqrt{\frac{n}{2j(k-1)}} = \sqrt{j(k-1)n/2}.$$

Example: Let $n = 100$, $j = 2$ and $k = 3$. We have $s = 3$ and $m' = 7$. We show some formula $f = f_1 \vee f_2$ in $\mathcal{C}'$ (for the mentioned parameters). The $\Pi's$ of f_1 are as follows.

$$\Pi_1 = v_1\bar{v}_2\bar{v}_3,\ \Pi_2 = v_4v_5\bar{v}_6,\ \Pi_3 = \bar{v}_7v_8v_9,\ \Pi_4 = v_{10}v_{11}v_{12},$$

$$\Pi_5 = \bar{v}_{13}\bar{v}_{14}\bar{v}_{15},\ \Pi_6 = v_{16}\bar{v}_{17}\bar{v}_{18},\ \text{and}\ \Pi_7 = \bar{v}_{19}\bar{v}_{20}v_{21}.$$

Therefore, f_1 is:

$$f_1 = \Pi_1 \vee \bar{v}_1\Pi_2 \vee \bar{v}_1\bar{v}_4\Pi_3 \vee v_2\bar{v}_4v_7\Pi_4 \vee v_2\bar{v}_5v_7\bar{v}_{10}\Pi_5 \vee v_3\bar{v}_5\bar{v}_8\bar{v}_{10}v_{13}\Pi_6 \vee v_3v_6\bar{v}_8\bar{v}_{11}v_{13}\bar{v}_{16}\Pi_7.$$

The subformula f_2 is defined over the variables $v_{22}, \ldots, v_{42}$.

The Adversary

The lower bound here holds when $jk = o(n/(\log n)^2)$ and the number of terms in the target formula is at most $\sqrt{j(k-1)n/2}$. We establish this bound using the techniques of Bshouty and Cleve [BC92].

Theorem 9 *For $jk = o(n/(\log n)^2)$, $k > 1$, and $m \leq \sqrt{j(k-1)n/2}$,*

$$\begin{aligned} \mathcal{E}_U(\textit{Read-}k\ \textit{Sat-}j\ \textit{DNF}_{n,m}) &\geq m \\ \mathcal{E}_U(\textit{Read-}k\ \textit{Sat-}j\ \textit{DNF}_n) &\geq m+1. \end{aligned}$$

Proof: We prove that the lower bound holds for the class C' defined above, by showing that the learner must ask at least m equivalence queries (or $m+1$ if m is no known a priori). Note that if m is less than the number of terms in the formulas in C', we drop all the extra terms from each formula.

Let f be the target formula in C'. The goal of the adversary is to ensure that each equivalence query (accompanied with a polynomial number of membership query) only helps the learner to know at most one term in f. If the number of terms in f is not known a priori, then the learner does not know when it has all the terms and needs one additional equivalence query.

The learner's task is to find the signs of the variables in each of the formula's Π's. Once this is done, the learner will be able to exactly identify the formula. Recall that the size of every Π is $s = \left\lfloor \sqrt{\frac{n}{2j(k-1)}} \right\rfloor$. Suppose the order of the literals in every Π is fixed. A vector $w \in \{0,1\}^s$ is the *sign vector* of Π if the ith bit of w gives the sign of the ith literal in Π. As stated above, the learner's task is to learn the sign vectors of the Π's.

Henceforth, we number the Π's of the target formula: $\Pi_1, \ldots, \Pi_m$. Let $\mathcal{P}_i \subseteq \{0,1\}^s$ be the set of sign vectors each of which is a candidate for being the sign vector of Π_i. In other words, every vector in $\mathcal{P}_i$ is consistent with the adversary's replies so far. At the beginning of the learning session the learner does not know the sign of any variable in any Π, so $\mathcal{P}_i = \{0,1\}^s$, for $i = 1, \ldots, m$. For $s = \left\lfloor \sqrt{\frac{n}{2j(k-1)}} \right\rfloor$, the initial size $\mathcal{P}_i$ is 2^s which is superpolynomial in n, provided that $jk = o(n/(\log n)^2)$. Later we will show that every membership query decreases the size of every $\mathcal{P}_i$ by at most 1. These two facts imply that a polynomial number of membership queries cannot decrease the size of any $\mathcal{P}_i$ to 1. In other words, a polynomial number of membership queries does not suffice to determine the vector sign of any Π.

The adversary's strategy in answering the membership and equivalence queries will be such that after e equivalence queries the learner knows only $\Pi_1, \ldots, \Pi_e$ but has gained no information about $\Pi_{e+1}, \ldots, \Pi_m$. Let π_i denote the sign vector of Π_i.

Let e be the number of equivalence queries that have been answered so far in the learning session. The adversary maintains the following invariants:

1. For $1 \leq \ell \leq e$, $\mathcal{P}_\ell$ contains exactly one element (the sign vector of Π_ℓ, as known to the learner). In addition, $\mathcal{P}_\ell$, $1 \leq \ell \leq e$ does not change in the future (so the adversary remains consistent).

2. For $\ell > e$, $|\mathcal{P}_\ell|$ is super-polynomial.

Let $g(x)$ be the disjunction of the first (known) e terms in the target formula f.

Answering membership queries. Suppose the learner asks MQ(a). The adversary answers as follows.

MQ1: $g(a) = 1$.

In this case, since $g(a) = 1$, it is the case that $f(a) = 1$, so the adversary answers 1. The learner has gained no information by this reply.

MQ2: $g(a) = 0$.

In this case the adversary answers 0, and updates the candidate sets $\mathcal{P}_{e+1}, \ldots, \mathcal{P}_m$ as follows. Recall that the length of the sign vector of each Π is s. Let b_i, $e < i \leq m$ be the ith block of size s in a. Note that a satisfies some Π_i if and only if the sign vector of Π_i is the complement vector of the block b_i. Therefore, in order to ensure that a falsifies all Π_i, $e < i \leq m$, we eliminate the complement of b_i from $\mathcal{P}_i$. Thus, we eliminate at most most one element from each $\mathcal{P}_i$, $e < i \leq m$.

Answering equivalence queries. For each equivalence query Equiv(h), the adversary answers as follows.

EQ1: $g \not\equiv h$, that is, there exists an assignment a such that $h(a) = 0$ and $g(a) = 1$.

In this case the adversary returns "no" accompanied with the counterexample a. The learner has gained no information because of this reply. (This case is similar to case **MQ1**.)

In order to maintain the invariants, the adversary picks an arbitrary element π in $\mathcal{P}_{e+1}$, and updates $\mathcal{P}_{e+1}$ to be exactly $\{\pi\}$.

EQ2: $h \not\equiv g$, that is, there exists an assignment a such that $h(a) = 1$ and $g(a) = 0$.

In this case the adversary answers "no" accompanied with a as a counterexample. In order to be consistent in future replies, the adversary updates the $\mathcal{P}_i$'s, $e < i \leq m$, as in case **MQ2**.

In addition, in order to maintain the invariants, the adversary picks an arbitrary element π in $\mathcal{P}_{e+1}$, and updates $\mathcal{P}_{e+1}$ to be exactly $\{\pi\}$.

EQ3: $g \equiv h$.

In this case, the adversary discloses a new term as follows. The adversary picks some element $\pi \in \mathcal{P}_{e+1}$, and returns the answer "no" accompanied with the counterexample a built in the following manner. The ith block, $1 \leq i \leq e$ is chosen to be the (unique) element in $\mathcal{P}_i$. The $(e+1)$st block is π, and the other blocks are fixed arbitrarily. The adversary updates $\mathcal{P}_{e+1}$ by setting it to be the complement of π. Observe that, by the way it was constructed, a falsifies g, so it falsifies h. However, the adversary disclosed the $(e+1)$st term and it is satisfied by a, so it satisfies the target formula.

Observe that an equivalence query discloses exactly one term from the target formula, so the first invariant is maintained. Observe also that as a result of a membership query or an equivalence query, the size of every $\mathcal{P}_i$, $e + 1 < i \leq m$, decreases by at most 1. Since the initial size of every $\mathcal{P}_i$ is super-polynomial, and since the learner is allowed only a polynomial number of membership

queries (and of course equivalence queries) the size of $\mathcal{P}_i$, $e+1 < i \leq m$, remain super-polynomial. So, the second invariant is maintained as well. □

6.3.2 The Case $k = 1$

By Theorem 7, the number of terms in a read-once sat-j DNF formula is at most j terms. We show a class $\mathcal{C}''$ of read-once sat-j DNF formulas, and show that the learner must ask at least j equivalence queries in order to identify the target formula. This bounds holds for $j = o(n/\log n)$.

Definition of the Target Class $\mathcal{C}''$

A formula $f \in \mathcal{C}''$ contains exactly j terms, each of size $s = \left\lfloor \frac{n}{j} \right\rfloor$. The ith term t_i, $1 \leq i \leq j$, is

$$t_i = \bigwedge_{p=(i-1)s+1}^{is} v_p^{c_p},$$

for some signs $c_{(i-1)s+1}, \ldots, c_{is}$.

Here, like in the previous lower bound, the learner's task is to identify the sign vector of every term. The adversary replies to the learner exactly like in the lower bound of the case $k > 1$. Note that since $j = o(n/\log n)$, the initial size of every candidate set is $2^s = 2^{\lfloor n/j \rfloor} = 2^{\omega(\log n)}$ which is super polynomial as claimed above.

We thus have proved:

Theorem 10 *For $j = o(n/\log n)$, $\mathcal{E}_U$ (Read-Once Sat-j DNF$_n$) is at least j if j is known a priori, and is at least $j+1$ otherwise.*

7 Monotone DNF Formulas

In this section we let Monotone DNF$_n$ represent the class of monotone DNF formulas on n variables, and we let Monotone DNF$_{n,m}$ be the subset of those formulas that have at most m terms. A learning algorithm is allowed time and membership queries polynomial in n and m (though for the former class m is not known to the learner a priori).

7.1 Lower Bounds

To prove lower bounds on the number of equivalence queries required to learn monotone DNF formulas, we prove the following key lemma demonstrating a trade off between membership and equivalence queries. The proof uses an adversary argument to show that for a certain subclass of monotone DNF formulas, membership queries reveal relatively little information.

Lemma 14 *$\mathcal{E}$ (Monotone $DNF_{n,m}, q) \geq m - d$ for any $0 < d < n$ satisfying $(\lfloor \frac{n}{d} \rfloor)^d > q + m - d$.*

Proof: For ease of exposition we consider the case where d divides n evenly. We prove the result holds for the following subclass of monotone DNF formulas. The target formula includes the following d terms that partition the variables into d blocks of size n/d (we call these the "fixed" terms, since we give them to the learner in advance).

$$\begin{aligned} T_1 &= v_1 v_2 \cdots v_{\frac{n}{d}} \\ T_2 &= v_{\frac{n}{d}+1} v_{\frac{n}{d}+2} \cdots v_{\frac{2n}{d}} \\ &\vdots \\ T_d &= v_{(\frac{d-1}{d})n+1} \cdots v_n. \end{aligned}$$

The remaining terms $\mathcal{T} = \{T_{d+1}, \ldots, T_m\}$ will each include all but one of the variables from each T_i with $i \leq d$ (so each such term contains $n-d$ variables). All the monotone DNF formulas obtainable in this fashion represent distinct functions. The task of the learner, then, is to decide whether each of the possible $(n/d)^d$ terms of the specified form are in the target formula.

Each time the learner makes a membership query on an instance $x \in \mathcal{X}$, the adversary replies:

$$f(x) = \begin{cases} 1 & \text{If } x \text{ satisfies one of } T_i, \ldots, T_d \text{ (or a previous query has stated } f(x) = 1). \\ 0 & \text{Otherwise.} \end{cases}$$

Thus when the adversary says $f(x) = 1$, the learner has obtained no new information. When the learner is told $f(x) = 0$, then x can satisfy at most one potential choice for a term in $\mathcal{T}$, so the learner's only new information is that one particular term is not in f.

The target formula contains up to $m - d$ initially unknown terms. The membership queries may eliminate up to q of the possible terms, but there are at least $(n/d)^d - q > m - d$ remaining terms about which the learner has no information. And $m - d$ of these terms may appear in f in any combination.

When the learner makes an equivalence query on some hypothesis h, the oracle replies "no" if it can return a counterexample of one of the following types:

1. $f(x) = 1$, for an example x that satisfies $T_1 \vee \cdots \vee T_d$, but has $h(x) = 0$.
2. $f(x) = 0$, for an example x that has two variables from some one of $T_1, \ldots, T_d$ set to 0, but has $h(x) = 1$.
3. $f(x) = 0$, for an example x that has exactly one variable from each of $T_1, \ldots, T_d$ set to 0 but has $h(x) = 1$ (and for which we haven't already stated that $f(x) = 1$).
4. $f(x) = 1$, for an example x that has exactly one variable from each of $T_1, \ldots, T_d$ set to 0 but has $h(x) = 0$ (and for which we haven't already stated that $f(x) = 0$).

In cases 1 and 2 the learner gains no new information. Case 3 is like the membership queries, where our negative counterexample eliminates just one particular term. In case 4, the learner has been able to discover a single new term from the target formula. Thus $m - d$ equivalence queries are required before the remaining terms of f can be discovered by the learner. □

We apply this lemma to prove lower bounds for both cases where m is known or unknown. We first consider the case in which m is a known input parameter for the learner.

Theorem 11

$$\mathcal{E}(\textit{Monotone } DNF_{n,m}) \geq m - \theta\left(\frac{\log m + \log n}{\log n - \log\log m}\right)$$

Proof: To prove this consequence of Lemma 14, we pick $d = \theta(\frac{\log m+\log n}{\log n-\log\log m})$ and let the number of membership queries q be an arbitrary polynomial function of m and n. To apply the lemma it suffices to show that for n sufficiently large, $(\frac{n}{d})^d > poly(m, n)$. Taking the logarithms of both sides, we get that the requirement is that $d(\log n - \log d) > O(\log n + \log m)$. If m is polynomial in n then this will hold if $d(\log n - \log d) > O(\log n)$ which is true since $d = \omega(1)$ and $d = o(n)$. Likewise if m is superpolynomial in n the given inequality holds. □

In the case where the learning algorithm is not given an a priori upper bound on the number of terms, we may prove a slightly stronger result.

Theorem 12 *For any* $0 < k_n < n - \omega(\log n)$ *with* $\lim_{n\to\infty} k_n = \infty$, *then for* n *sufficiently large*

$$\mathcal{E}(\textit{Monotone } DNF_n) \geq m - k_n$$

Proof: Pick $d = k_n$. If at any point after having made e equivalence queries the algorithm has made a number of membership queries superpolynomial in n and e (answered by the strategy above), the adversary decides there is only one more term in f, which means the algorithm has made superpolynomial number of membership queries. Thus the algorithm can only ever make a number of membership queries polynomial in n. The result follows since $(n/k_n)^{k_n}$ grows superpolynomially. □

7.2 Upper Bounds

In this section we describe an algorithm that matches the above lower bounds. We begin by briefly describing Angluin's algorithm [Ang88] for learning a monotone DNF formula using at most m equivalence queries. A *prime implicant* of a Boolean formula f is a conjunction t (not containing contradictory literals) such that t implies f, but no proper subset of t implies f. For general DNF formulas the number of prime implicants may be exponentially larger than the number of terms. But for monotone DNF formulas, the number of prime implicants is bounded above by the number of terms in the formula. Furthermore the prime implicants of a monotone function include no negated variables.

Learn-Monotone-DNF(n, k)

Initialize the formula h to have no terms (i.e. be everywhere false).
Repeat
    Find a positive counterexample x as follows.
        If h contains at most k terms, then search exhaustively for an x with
          at most k bits set to 0 such that $h(x) = 0$ and $f(x) = 1$.
          (Return h if none is found.)
        Otherwise (h contains more than k terms),
          Set x to Equiv(h) (or return h if the answer is "yes").
    Greedily flip as many bits as possible to 0 in x, while preserving $f(x) = 1$.
    When no further changes are possible,
        add the conjunction of variables still set to 1 in x as a new term for h.

Figure 4: An algorithm to learn monotone DNF with $m - k$ equivalence queries and $O\left(k\left(\frac{n}{k}\right)^k + mn^2\right)$ membership queries.

Given this observation, there is a fairly straightforward exact identification algorithm due to Angluin [Ang88] (based on a previous PAC learning algorithm of Valiant [Val84]). We use each equivalence query to find a new prime implicant. Our current hypothesis is the disjunction of all known prime implicants (initially the always false hypothesis). Then each counterexample x can be used to find a new prime implicant by walking it towards the all zeros example (using membership queries to decide which variables should be set to 0). It is easy to see that the resulting example will satisfy exactly the variables of some new prime implicant. This technique requires $m + 1$ equivalence queries and mn membership queries.

A simple optimization allows us to find the first prime implicant without making an equivalence query. Monotonicity implies that if the target formula is not identically 0 then $f(1_n) = 1$ (1_n is the all 1's example). This can be used to find the first term, reducing our equivalence query requirement to m. That observation gives us the special case (for $k = 0$) of an algorithm we present in Figure 4. This new algorithm can reduce the number of equivalence queries by an arbitrary number k. This savings is at a cost of time and membership queries exponential in k, but this will be enough to show that our previous lower bounds are tight.

Theorem 13 *There is an exact identification algorithm for monotone DNF formulas that takes as input n and a non-negative integer $k < m$, and learns the target formula using $m - k$ equivalence queries and $O\left(k\left(\frac{n}{k}\right)^k + mn^2\right)$ time and membership queries.*

Proof: This algorithm (shown in Figure 4) finds $k + 1$ prime implicants of f before making any equivalence queries. The key observation is that as long as we have discovered at most k prime

implicants, then if there is any counterexample there will be one that has only k variables set to 0 (and we can exhaustively test all possible such counterexamples, of which there are fewer than $(en/k)^k$). This is because any positive counterexample fails to satisfy our k prime implicants. Given that such a counterexample exists, there is some set of k or fewer variables covering our prime implicants that are set to 0 in the counterexample, and given that those variables are 0 in some positive counterexample, the example that has only those variables set to 0 will still be both a positive example and a counterexample. Thus for the first k terms, we use brute force enumeration to find counterexamples. After this we use $m-k$ equivalence queries to learn the remaining $m-k-1$ terms in the standard manner. □

Based on this technique we prove two upper bounds for learning monotone DNF formulas. In the case where m is not known, the learner needs $m - \Theta(1)$ equivalence queries. When m is known, we prove that the number of queries is reduced to $m - \Theta(\frac{\log m + \log n}{\log n - \log\log m})$. Note that these bounds differ only when m is superpolynomial in n. They both follow from Theorem 13 by substituting the appropriate quantities for k.

Corollary 14 *For any constant $c > 0$, $\mathcal{E}(\text{Monotone } DNF_n) \leq m - c$.*

Corollary 15 *For any constant $c > 0$, $\mathcal{E}(\text{Monotone } DNF_{n,m}) \leq m - c\left(\frac{\log m + \log n}{\log n - \log\log m}\right)$.*

8 Horn Sentences (and DNF)

In this section we let Horn Sentence$_n$ represent the set of Horn sentences over n variables, and we let Horn Sentence$_{n,m}$ be the subset of those formulas that have at most m clauses. From corollary 2 we have an upper bound on the number of equivalence queries needed when computation time is unlimited, but only a polynomial number of membership queries are allowed. In this section we show a matching lower bound that shows that no fewer equivalence queries (modulo big-Oh notation) will suffice unless there are a super-polynomial number of membership queries. The question of whether this lower bound can be achieved by a computationally efficient algorithm remains open.

8.1 Lower Bound

Suppose that for some $c > 0$ there is an algorithm that learns the class of Horn sentences in time less than $(mn)^{c+1}$. In this section we prove our lower bound by considering the following subclass of Horn Sentences. Let $d = \lceil (c+1)(\log n + \log m) \rceil$ and $q = \lfloor n/2d \rfloor$. Divide the $2dq$ variables $v_1, \ldots, v_{2dq}$ into q blocks each of which contains $2d$ variables. Specifically, for $1 \leq i \leq q$, block B_i will contain variables $v_{2d(i-1)+1}, \ldots, v_{2di}$. Given a vector x we use $x[B_i]$ to denote the portion of x that corresponds to block B_i. That is, $x[B_i]$ contains $x_{2d(i-1)+1}, \ldots, x_{2di}$.

For each of the q blocks of variables we will construct a Horn sentence in the following manner. Let $y_1, \ldots, y_{2d}$ be the $2d$ variables in block B_i. We define[4]

$$P_i = \bigwedge_{j=1}^{d} (y_{2j-1}y_{2j} \rightarrow y_{(2j+1) \bmod 2d}) \wedge (y_{2j-1}y_{2j} \rightarrow y_{(2j+2) \bmod 2d}).$$

So for example if $d = 3$ we have

$$P_i = (y_1y_2 \rightarrow y_3)(y_1y_2 \rightarrow y_4) \cdots (y_5y_6 \rightarrow y_1)(y_5y_6 \rightarrow y_2).$$

Observation 1 *For $1 \leq i \leq q$, $1 \leq j \leq d$, P_i has the property that if both variables in any pair y_{2j-1}, y_{2j} are 1 then it will be false unless* all *$2d$ variables are 1.*

Let $\mathcal{S}^{(dq)} \subset \{0,1\}^{2dq}$ be the set of bit strings for which each consecutive disjoint pair consists of a 1 and 0. That is: $\mathcal{S}^{(dq)} = \{(s_1, \ldots, s_{2dq}) \mid (s_{2j-1}, s_{2j})$ is $(0,1)$ or $(1,0)$ for $1 \leq j \leq dq\}$. For any vector $s \in \mathcal{S}^{(dq)}$ define $I(s) = \{j \mid s_j = 1\}$, and for each $s \in \mathcal{S}^{(dq)}$ let

$$R_s = \left(\bigwedge_{j \in I(s)} v_j \right) \rightarrow 0.$$

Observation 2 *For any $x \in \{0,1\}^{2dq}$ and $s \in \mathcal{S}^{(dq)}$, $R_s(x) = 0$ if and only if $x_j = 1$ for all $j \in I(s)$.*

Finally, for $s_1, \ldots, s_t \in \mathcal{S}^{(dq)}$, let $F_{s_1,\ldots,s_t} = P_1 \wedge \cdots \wedge P_q \wedge R_{s_1} \wedge \cdots \wedge R_{s_t}$, and let

$$\mathcal{C} = \left\{ F_{s_1,\ldots,s_t} \mid s_1, \ldots, s_t \text{ are in } \mathcal{S}^{(dq)} \text{ and are distinct } \right\}.$$

Theorem 16 *For $m - n = \Omega(m)$, $\mathcal{E}(\text{HornSentences}_{n,m}) = \Omega\left(\frac{mn}{\log n + \log m}\right)$.*

Proof: We prove that the above lower bound holds for the class $\mathcal{C}$ defined above. Since the number of clauses in each P_i is $2d$, there are $2dq < n$ clauses in $P_1 \wedge \cdots \wedge P_q$ so fix $t = m - 2dq$. Since

$$tq = (m - 2dq)q \geq (m - n)q = \Omega\left(\frac{mn}{\log n + \log m}\right),$$

the desired result will follow if the adversary can force the learner to make tq equivalence queries before obtaining exact identification.

Let f be the target function. Observe that the learner knows $P_1 \wedge \cdots \wedge P_q$ before the learning session begins. The goal of the adversary is to ensure that each equivalence query (combined with a polynomial number of membership queries) will only help the learner to determine one block of some s_i (i.e. one of $s_i[B_1], \ldots s_i[B_q]$). Since there are tq such blocks, once this goal is achieved the result will follow.

[4] Although P_i is a formula defined over the $2d$ variables in block B_i, we use $P_i(x)$ to denote $P_i(x[B_i])$.

For ease of exposition, we further divide $s_1, \ldots s_t$ each into q blocks each containing $2d$ bits. We denote these blocks by $b_1, \ldots, b_q, b_{q+1}, \ldots, b_{(t-1)q}, b_{(t-1)q+1}, \ldots, b_{tq}$. The adversary's strategy in answering the membership and equivalence queries will be such that after e equivalence queries the learner will know only $b_1, \ldots, b_e$ but has gained no information about $b_{e+1}, \ldots, b_{tq}$. We say that b_ℓ is *known* if $\ell \leq e$ and *unknown* if $\ell > e$.

Let $D_\ell^{(e)}$ denote the values for b_ℓ that are consistent with all examples seen by the learner after e equivalence queries have been answered. During the proof we will often focus on the elements of $D_\ell^{(e)}$ that are in block i of some s_j. Thus for $1 \leq i \leq q$, let

$$\mathcal{D}_{B_i}^{(e)} = \{D_\ell^{(e)} \mid \ell = (j-1)q + i \text{ for } 1 \leq j \leq t\}.$$

Note that for all j, $D_j^{(0)} = \mathcal{S}^{(d)}$ and thus $|D_j^{(0)}| = 2^d \geq (mn)^{c+1}$ at the beginning of the learning session.

Let e be the number of equivalence queries that have been answered so far in the learning session. The adversary will maintain the following invariants.

1. For $1 \leq \ell \leq e$, $D_\ell^{(e)} = \{b_\ell\}$. That is, $b_1, \ldots, b_e$ are known.
2. For $1 \leq \ell_1 < \ell_2 \leq e$, $D_{\ell_1}^{(e)} \cap D_{\ell_2}^{(e)} = \emptyset$. That is, $b_1, \ldots, b_e$ are disjoint.
3. For $\ell > e$, $\left(D_1^{(e)} \cup \cdots \cup D_e^{(e)}\right) \cap D_\ell^{(e)} = \emptyset$. That is, $b_1, \ldots, b_e$ are not included in the set of candidates for $b_{e+1}, \ldots, b_{tq}$.
4. For $D_1, D_2 \in \mathcal{D}_{B_i}^{(e)}$ such that $|D_1| > 1$ and $|D_2| > 1$, $D_1 = D_2$. That is, all unknown values in a given block have the same set of candidates remaining.
5. For any l, if $|D_\ell^{(e)}| = 1$ then $D_\ell^{(w)} = D_\ell^{(e)}$ for $w > e$. That is, once b_ℓ is known $\mathcal{D}_\ell$ does not change.
6. Let Q_e be the number of membership and equivalence queries asked by the learner up to (and including) the e^{th} equivalence query. Then

$$|D_\ell^{(e)}| \geq (mn)^{c+1} - Q_e \text{ for } \ell > e.$$

Notice that since $Q_e < 2(mn)^c$, it follows that $|D_\ell^{(e)}| \geq (mn)^c$ for $\ell > e$. We now define the strategy that will be used by the adversary to respond to the queries. Each query will enable the learner to determine only one of the tq blocks $b_1, \ldots, b_{tq}$ and further can eliminate at most one element from each $D_\ell^{(e)}$ for $\ell > e$. Thus adversary can force tq equivalence queries as desired.

After e equivalence queries have been answered, $r = \lfloor e/q \rfloor$ is the largest j such that s_j is completely known, and $p = e - qr$ is the index of the last known block within s_{r+1}. Let I_e be the

indices of the elements of s_{r+1} that are known to be 1. That is $I_e = \{j \mid j \in I(s_{r+1}[B_i])$ for $1 \leq i \leq p.\}$ Now let

$$R_e^* = \left(\bigwedge_{j \in I_e} v_j\right) \wedge \left(\bigwedge_{j=2dp+1}^{2dq} v_j\right) \rightarrow 0.$$

Thus R_e^* contains all variables whose corresponding indices in s_{r+1} are known to be 1 and all variables corresponding to the unknown elements in s_{r+1}.

Observation 3 *The antecedent of R_e^* is a superset of the antecedent of $R_{s_{r+1}}$ and thus $R_e^*(x) = 0$ implies that $R_{s_{r+1}}(x) = 0$.*

Let $g_e(x) = P_1(x) \wedge \cdots \wedge P_q(x) \wedge R_{s_1}(x) \wedge \cdots \wedge R_{s_r}(x) \wedge R_e^*(x)$. Applying Observation 3 it follows that for $1 \leq e \leq tq$, if $g_e(x) = 0$ then $f(x) = 0$.

Answering a Membership Query. For each membership query, MQ(a), the adversary responds as follows.

Case MQ1: $g_e(a) = 0$.

In this case the adversary replies 0. Since $g_e(a) = 0$ implies $f(a) = 0$ no information is given to the learner by this answer.

Case MQ2: $g_e(a) = 1$ and there exists $i \in \{1, \ldots, dq\}$ such that $(a_{2i-1}, a_{2i}) = (0, 0)$.

In this case the adversary returns 1. Since $(a_{2i-1}, a_{2i}) = (0, 0)$ it follows that $R_s(a) = 1$ for any s, and thus no information is given to the learner by this answer.

Case MQ3: $g_e(a) = 1$ and for all $i \in \{1, \ldots, dq\}$, $(a_{2i-1}, a_{2i}) \neq (0, 0)$.

Since $P_i(a[B_i]) = 1$ for all blocks i, by Observation 1 we know that $a[B_i]$ is either all 1's or an element of $\mathcal{S}^{(d)}$. If $a[B_i]$ contained all 1s for $1 \leq i \leq q$, it would follow that $R_e^*(a) = 0$. Thus, there exists an i_0 such that $a[B_{i_0}] \in \mathcal{S}^{(d)}$. The adversary returns 1 and removes $a[B_{i_0}]$ from the set of candidates for all blocks b_ℓ that are not known and correspond to B_{i_0}. That is for all $D \in \mathcal{D}_{B_{i_0}}^{(e)}$ such that $|D| > 1$, update $D \leftarrow D \setminus a[B_{i_0}]$. Note that after this update if each unknown r_j is selected from its associated D then we are assured that $g_e(x) = 1$ for all vectors x.

Answering a Equivalence Query. For each equivalence query, Equiv(h), the adversary responds as follows.

Case EQ1: There exists a vector a such that $h(a) = 1$ and $g_e(a) = 0$.

The adversary will handle this situation just as it did in Case MQ1 where the learner asked the membership query MQ(a) for which $g_e(a) = 0$. Finally, to maintain the invariants, the adversary selects an arbitrary $u \in D_{e+1}^{(e)}$ and sets $D_{e+1}^{(e+1)} \leftarrow u$. Let $p = e - q\lfloor e/q \rfloor$. For each $D_\ell^{(e)} \in \mathcal{D}_{B_p}^{(e)}$ such that $|D_\ell^{(e)}| > 1$, the adversary sets $D_\ell^{(e+1)} \leftarrow D_\ell^{(e)} \setminus \{u\}$. Also e is incremented in all other $D_\ell^{(e)}$.

Case EQ2: There exists a vector a such that $h(a) = 0$ and $g_e(a) = 1$.

The adversary will handle this situation just as it did in Cases MQ2 and MQ3 where the learner asked the membership query MQ(a) for which $g_e(a) = 1$. As in Case EQ1, the adversary then updates the candidate sets to maintain the invariants.

Case EQ3: $h \equiv g_e$.

In this case we will take advantage of the fact for all ℓ, $D_\ell^{(e)}$ satisfies the invariants. Observe that all updates made in the above cases preserve these invariants. By invariant 4 it follows that for all $D_1, D_2 \in \mathcal{D}_{B_p}^{(e)}$ for which $|D_1| > 1$ and $|D_2| > 1$, $D_1 = D_2$ where p is the block number of b_e. That is, b_e corresponds to $s_{r+1}[B_p]$ where $r = \lfloor e/q \rfloor$. For any such $D \in \mathcal{D}_{B_p}^{(e)}$ for which $|D| > 1$ select some $u \in D$ and set $s_{r+1}[B_p] = u$. Consider the example x in which $x[B_i] = 1_{2d}$ for $i \neq p$ and $x[B_p] = u$. By Invariant 3, $u \notin D_j^{(e)}$ for $j < e$, and thus it follows that $h(x) = 1$. Since $R_{e+1}^*(x) = 0$ it follows that $g_{e+1}(x) = 0$ and thus x can be returned as the counterexample. Finally, as in Case EQ1, the adversary updates the candidate sets to maintain the invariants.

□

9 Read-Once Formulas Over Various Bases

In this section we prove an upper bound on the number of equivalence queries needed to identify read-once formulas. This is achieved as a consequence of a more general result, showing that an algorithm that makes use of equivalence queries only to generate justifying assignments (defined below) needs to make only $O(n/\log n)$ queries. This is an improvement from a previous technique that uses n queries [AHK89, BHHK91], and immediately gives us improved upper bounds for various classes of read-once formulas and non-monotone switch configurations. These upper bounds are tight from the work of Bshouty and Cleve [BC92].

In this section we consider the following classes of read-once formulas. Let $\mathrm{ROF}_n(B)$ denote the set of read once-formulas whose gates are labeled with functions from B (the "basis"). We let B_k denote the basis of all boolean functions over k inputs, for a constant k. Let Switch Configurations$_n$ denote the set of n element switch configurations (in the general non-monotone case where the sign of each switch is not known a priori). Let $\mathrm{AROF}_n^{(\mathcal{F})}(+, \times, /, -)$ denote the class of n variable *arithmetic* read-once formulas over the basis of addition, subtraction, multiplication, and division a the field $\mathcal{F}$ (for this non-Boolean class, the inputs are variables or constants from $\mathcal{F}$, and the output is a value in $\mathcal{F} \cup \{\infty, 0/0\}$).

It follows from the work of Bshouty and Cleve [BC92] that

$$\mathcal{E}(\mathrm{ROF}_n(AND, OR, NOT)) = \Omega\left(\frac{n}{\log n}\right)$$

$$\begin{aligned}
\mathcal{E}\left(\mathrm{ROF}_n(B_k)\right) &= \Omega\left(\frac{n}{\log n}\right), \\
\mathcal{E}\left(\text{Switch Configurations}_n\right) &= \Omega\left(\frac{n}{\log n}\right) \\
\mathcal{E}\left(\mathrm{AROF}_n^{(\mathcal{F})}(+,\times,/,-)\right) &= \Omega\left(\frac{n\log|\mathcal{F}|}{\log n}\right), \text{ when } |\mathcal{F}| = o(n/log n).
\end{aligned}$$

9.1 Generating Justifying Assignments With $O(n/\log n)$ Equivalence Queries

We now describe a technique of generating justifying assignments with $O(n/\log n)$ equivalence queries that can then be used to get algorithms that match the above lower bounds. We start with few definitions. A class $\mathcal{C}$ is *closed under zero projection* if for any function $f \in \mathcal{C}$, fixing some variables of f to 0 produces a function still in $\mathcal{C}$. A *justifying assignment* for an input variable is an instance whose classification changes if the value of the variable is changed. Among other things, the justifying assignment is a witness to the fact that the given variable is relevant.

Define the vector l_m to be $(\overbrace{l,\ldots,l}^{m})$ where $l \in \{0,1,\star\}$. For an input vector $x = (x_1,\ldots,x_n)$ and a set of variables $V = \{v_{i_1},\ldots,v_{i_k}\}$, we denote $x(V) = (x_{i_1},\ldots,x_{i_k})$ (i.e. the input vector in the lower dimensional space induced by the variables V). A partial assignment is an input setting that assigns $\star$ to some of the variables (to indicate the variable is unassigned). For a partial assignment p and an assignment a, we will denote by $p|a$ the assignment that replaces the stars of p with the corresponding values in a. For a partial assignment p and a Boolean function f we define $f_p(a) = f(p|a)$. For an assignment a and a variable v the assignment $b = a_{\neg v}$ is the assignment that satisfies $b(v) = \neg a(v)$ and $b(v') = a(v')$ for any variable $v' \neq v$.

Given two assignments a and b such that $f(a) \neq f(b)$, the procedure **Walk** a towards b is a procedure that continues to flip bits in a that are different from b, while keeping $f(a) \neq f(b)$. The procedure generates a new assignment a' such that for any variable v, if $a'(v) \neq b(v)$ then $f(a'_{\neg v}) \neq f(a')$.

We now describe that standard transformation to produce a set of justifying assignments using n equivalence queries. Suppose we have justifying assignments for some subset Y of the variables (initially empty), and suppose those justifying assignments all agree on setting the variables in $V \setminus Y$ to 0. Then if p is the partial assignment that sets to $\star$ all variables in Y and to 0 all variables in $V \setminus Y$, we can use the membership and justifying assignment algorithm for $\mathcal{C}$ to learn a hypothesis h equivalent to f_p (the condition that $\mathcal{C}$ is closed under zero projections implies that f_p is in $\mathcal{C}$). We make an equivalence query on h. If we get a counterexample y then $h(y) = f_p(y) \neq f(y)$, and we can use **Walk** y towards $p|y$ to find a justifying assignment for one or more new variables. We then repeat with a p that assigns values to strictly fewer variables, and when $p = \star_n$ we are done.

We now present an improved transformation that finds at least $\Omega(\log n)$ new variables with each equivalence query. Recall that an $(n,k)-$universal set is a set $\{b_1,\ldots,b_t\} \subseteq \{0,1\}^n$ such that every

subset of k variables assumes all of its 2^k possible assignments in the b_i's.

Theorem 17 *Let C be a class that is closed under zero projections If C is learnable in polynomial time from $M(n)$ membership queries, given justifying assignments for all the relevant variables, then for any $\epsilon > 0$ there is a $q = O(n^{1+\epsilon}M(n) + n^3)$ such that*

$$\mathcal{E}(C, q) \leq \frac{n}{\lfloor(\epsilon/4)\log n\rfloor}.$$

Proof: The algorithm for this reduction is shown in Figure 5. As before, there is a main loop where each iteration begins by running the membership query and justifying assignment subroutine to learn an $h \equiv f_p$ for the p that assigns 0 to the variables in $V \setminus Y$, using known justifying assignments for the variables in Y. But before we ask the equivalence query $h \equiv f$, we also learn a family of f_{p_i}'s determined by an $(n, \lfloor(\epsilon/4)\log n\rfloor)$-universal set of size $t \leq n^c$. We define f_{p_i} (for $i = 1$ to t) to be the partial assignment that sets the variables in $V \setminus Y$ as in the i'th element of the universal set. In other words, every possible assignment of values to some subset of $\lfloor(\epsilon/4)\log n\rfloor$ variables from $V \setminus Y$ is realized by some f_{p_i}. To learn each f_{p_i}, we test whether $f_p(x) = f_{p_i}(x)$ for all justifying assignments in A and for all the membership queries made by the justifying assignment algorithm when learning f_p. If this is the case then that algorithm outputs the same hypothesis for both target functions, and the correctness of the algorithm implies that $f_p \equiv f_{p_i}$. If however we find some $f_p(x) \neq f_{p_i}(x)$ this implies (by definition) that $f(p|x) \neq f(p_i|x)$, and since those examples agree on all variables in Y, using **Walk** we can find a new justifying assignment without making any equivalence queries.

Now we argue that if all f_{p_i}'s are equivalent to f_p, and we make an equivalence query on h, then we shall be able to find $\Omega(\log n)$ new justifying assignments from the counterexample. We start by walking our counterexample y towards $p|y$, to give us at least one new justifying assignment. After this walk we will have y' for which $f(y') \neq f_p(y')$, and y' is a justifying assignment for all the variables $Y_1 \subset V \setminus Y$ on which y' differs from $p|y'$. If $|Y_1| \geq \lfloor(\epsilon/4)\log n\rfloor$, we are done. If not, then there is some p_i that agrees with y' on all the variables in Y_1. But we know $f_{p_i} \equiv f_p$, so $f(y') \neq f_{p_i}(y')$. We now **Walk** y' towards $p_i|y'$, and we are guaranteed that we find justifying assignments for variables in $V \setminus (Y \cup Y_1)$. Call these variables Y_2. If $|Y_1| + |Y_2| \geq \lfloor(\epsilon/4)\log n\rfloor$, again, we are done. If not we can repeat with a different p_i that agrees with y' on $Y_1 \cup Y_2$, and so on. The bounds in the statement of the theorem follow from straightforward analysis. □

Applying the above technique to previous algorithms [AHK89, BHH92a, BHHK91, RS90] we obtain the following result.

Corollary 18 *The quantities $\mathcal{E}(ROF_n(AND, OR, NOT))$, $\mathcal{E}(ROF_n(B_k))$, and $\mathcal{E}(Switch\ Configurations_n)$ are all $O(n/\log n)$.*

In all these cases the transformation adds a factor of $O(n^{1+\epsilon})$ membership queries and running time to the original algorithm and saves a factor of $\lfloor(\epsilon/4)\log n\rfloor$ equivalence queries.

Find-JA-Loop(n)

Initialize $p = 0_n$, $Y = \emptyset$, $A = \emptyset$. Let $\{b_1, \ldots, b_t\}$ be a $(n, \lfloor(\epsilon/4)\log n\rfloor)$-universal set of size $t \leq n^\epsilon$.
Repeat
  Learn a hypothesis $h \equiv f_p$ (using membership queries and the justifying assignments A for the relevant variables Y).
  For $i = 1$ to t define the partial assignment p_i as follows

$$(p_i)_j = \begin{cases} \star & v_j \in Y \\ (b_i)_j & \text{Otherwise.} \end{cases}$$

  Test each f_{p_i} on all justifying assignments in A and on all the membership queries made by the algorithm to learn f_p.
    If on some point x, some $f_{p_i}(x) \neq f_p(x)$
      Walk $p_i|x$ towards $p|x$ to find one or more new justifying assignments.
      Update Y and A, and iterate (having made no equivalence queries).
    Otherwise, continue (all $f_{p_i} \equiv f_p$).
  Perform Equiv(h)
    If the answer is "yes", output h and halt.
    Otherwise let y be the counterexample.
  Repeat till this loop has added $\lfloor \frac{\epsilon}{4} \log n \rfloor$ new variables to Y
    Pick an i for which p_i and y agree on all the new variables added so far
      (we know $f(y) \neq h(y) = f_p(y) = f_{p_i}(y)$).
    Walk y towards $p_i|y$ to find new justifying assignments. Update Y and A.

Figure 5: An algorithm to use only $n/\log n$ equivalence queries to generate justifying assignments.

9.2 Arithmetic Read-once Formulas

We now consider the class of arithmetic read-once formulas $\mathrm{AROF}_n^{(\mathcal{F})}(+,\times,/,-)$. There is a polynomial time identification algorithm for this class that uses membership queries and n equivalence queries [BHH92a]. When the size of the field $\mathcal{F}$ is at least $2n+5$ then the algorithm does not use equivalence queries at all (however, the algorithm is randomized in this case). The lower bound, established by Bshouty and Cleve [BC92], of $\Omega(n \log |\mathcal{F}| / \log n)$ on the number of equivalence queries holds when the size of $\mathcal{F}$ is $o(n/\log n)$. It is an open problem whether equivalence queries are essential when the size of $\mathcal{F}$ falls in the gap between $\Theta(n)$ and $\Theta(n/\log n)$. The tight bound on the number of equivalence queries proved here is when the size of $\mathcal{F}$ is $o(n/\log n)$. The algorithm uses equivalence queries only to generate justifying assignments, but it is not immediately obvious that we can apply our techniques because of the difficulty of non-boolean variables.

We need to make only a slight change in our algorithm **Find-JA-Loop** to make it work for arithmetic read-once formulas. We want the universal set generated in the first step in the algorithm to be over all values in $\mathcal{F}$. That is, every subset of k variables assumes all its $|\mathcal{F}|^k$ values in the universal set. Having made this change, the algorithm **Find-JA-Loop** learns the target arithmetic read-once formula.

Instead of dealing with a universal set that contains values from $\mathcal{F}$, we work with a universal set that contains only 0's and 1's, in which the values of $\mathcal{F}$ are represented in binary. Since we want the universal set to contain all values in $\mathcal{F}$, we need $\log |\mathcal{F}|$ bits to represent every value. We look at the columns of the set as being divided into blocks of $\log |\mathcal{F}|$ columns each, each block corresponds to a value in $\mathcal{F}$. The number of columns needed in the universal set is therefore $n \log |\mathcal{F}|$. We want that every subset of size k of the variables assumes all its possible field values, so we require that every $k \log |\mathcal{F}|$ columns in the universal set assume all the possible (binary) values. Thus, the universal set needed is an $(n \log |\mathcal{F}|, k \log |\mathcal{F}|)$–universal set, and its size is

$$2^{3k \log |\mathcal{F}|} \log(n \log |\mathcal{F}|).$$

We want this quantity to be polynomial in n, so:

$$2^{3k \log |\mathcal{F}|} \log(n \log |\mathcal{F}|) \leq n^c,$$

for some constant c. Taking the logarithm of both size, and canceling small terms, we get that k must satisfy:

$$k = \frac{c \log n}{\log |\mathcal{F}|}.$$

Using this k, every iteration of the **Find-JA-Loop** finds justifying assignments for k new variables. The number of iterations (or, equivalently, the number of equivalence queries) is at most

$$\frac{n}{k} = \frac{n \log |\mathcal{F}|}{c \log n},$$

which matches the lower bound.

One last remark is that after building the $(n \log |\mathcal{F}|, k \log |\mathcal{F}|)$−universal set, we go over the set, translating the strings in every block to values of $\mathcal{F}$ (this would make the rest of the algorithm cleaner).

Corollary 19 *For any field $\mathcal{F}$, $\mathcal{E}\left(AROF_n^{(\mathcal{F})}(+,\times,/,-)\right)$ is $O(n \log |\mathcal{F}| / \log n)$.*

10 Deterministic Finite State Automata

In this section we present asymptotically tight bounds on the number of equivalence queries needed to learn DFAs. Let n denote the number of states in a minimum-state DFA that represents the target regular language U, and let $k = |\Sigma|$. For $s_1, s_2 \in \Sigma$, we denote the concatenation of s_1 and s_2 by $s_1 \cdot s_2$. To distinguish between the situations in which n is known or unknown to the learner, let $DFA_{n,\Sigma}$ denote the case when n is known and DFA_Σ denote the case in which n is not known.

Given that there is a polynomial time algorithm that can determine if two DFAs are equivalent and in addition outputs a minimum length counterexample if they are not equivalent, it seems reasonable to assume that the equivalence oracle uses such an example. Thus the results presented here all assume that the counterexamples returned by the equivalence oracle have length at most n. These results can easily be generalized to have the complexity depend on the length of the longest counterexample received by the learner.

10.1 Lower Bound

In this section we present our lower bound. This lower bound holds even when there is no restriction on the time or hypothesis class used by the algorithm and the learner knows n a priori.

Theorem 20 *For any constant $c \geq 1$, $\mathcal{E}\left(DFA_{n,\Sigma}, n^c - 1\right) \geq \left\lfloor \frac{n-2}{c \log_k n} \right\rfloor$.*

Proof: Let L be an arbitrary string of $n-2$ elements from Σ. We let $L[i]$ denote the ith element of L. For ease of exposition when defining the transition function, let $L[n-1]$ denote some special symbol that is not in Σ. Let $A \subseteq \{1, 2, \ldots, \left\lfloor \frac{n-2}{c \log_k n} \right\rfloor\}$. The adversary will select L and A as the learning session progresses. The target regular language, U, is defined by the following n-state DFA. The state set, $Q = \{0, \ldots, n-1\}$, the initial state $q_0 = 1$, accepting state set $F = \{i(c \log_k n)+1 \mid i \in A\}$, and the transition function δ is defined as:

$$\begin{aligned} \delta(0, \sigma) &= 0, \text{ for all } \sigma \in \Sigma \text{ (so state 0 is a dead state)} \\ \delta(q, \sigma) &= 0, \text{ for all } q \in Q \text{ and } \sigma \neq L[q] \\ \delta(q, \sigma) &= q + 1 \text{ for all } q \in Q \text{ and } \sigma = L[q]. \end{aligned}$$

The adversary will respond "no" to every membership query asked by the learner. We now describe how the adversary can respond to the ith equivalence query, h_i. Observe that since the learner

makes less than n^c membership queries, there exist some string L_i from Σ^* of length $c\log_k n$ that does not appear in the ith set of $c\log_k n$ positions of any strings given to the membership oracle. The adversary then uses L_i as the ith block of $c\log_k n$ characters of L. Finally, we want to classify the string $L_1 \cdot L_2 \cdots L_i$ so that it is a counterexample to h_i. Observe that the target DFA is designed so that on input $L_1 \cdots L_i$, it ends in state $q_i = 1 + i(c\log_k n)$. Thus the adversary can just place i into A if and only if h_i does not accept $L_1 \cdots L_i$. Then $L_1 \cdots L_i$ can be returned as the counterexample to the equivalence query.

The above argument can be applied as long as $i \leq \left\lfloor \frac{n-2}{c\log_k n} \right\rfloor$, giving the stated lower bound. □

10.2 Upper Bound

We now provide an efficient algorithm that matches the lower bound of the previous section. For the upper bound we use a modification of Schapire's algorithm [Sc91] which is itself a modification of Angluin's original algorithm [Ang87b].

For $x \in \Sigma^*$, we define $\gamma(x) = 1$ if and only if $x \in U$. An observation table [Ang87b], denoted by (S, E, T), records the value of $\gamma(x)$ for $x \in (S \cup S \cdot \Sigma) \cdot E$. We denote the row of table (S, E, T) labeled by $s \in S \cup S \cdot \Sigma$ by row(s). We say that (S, E, T) is *closed* if for all $s' \in S \cdot \Sigma$, there exists an $s \in S$ such that row(s') = row(s). We say that (S, E, T) is *consistent* if whenever $s_1, s_2 \in S$ satisfy row(s_1) = row(s_2), then for all $a \in \Sigma$, row$(s_1 \cdot a)$ = row$(s_2 \cdot a)$. Given a closed, consistent observation table (S, E, T), we define $M(S, E, T)$ as the DFA over alphabet Σ, with state set $Q = \{\text{row}(s) : s \in S\}$, initial state $q_0 = \text{row}(\lambda)$, accepting state set $F = \{\text{row}(s) : s \in S, \gamma(s) = 1\}$ and transition function $\delta(\text{row}(s), a) = \text{row}(s \cdot a)$. Angluin has shown that $M(S, E, T)$ is a well defined minimum-state DFA that correctly classifies all strings represented in (S, E, T). Finally, for DFA M and string $x \in \Sigma^*$, we define $\gamma_M(x) = 1$ if and only if x is accepted by M.

We now describe our algorithm. Initially S and E are initialized to the set $\{\lambda\}$. Using membership queries the learner fills in the entries of T to make it closed. (We never place $s_1, s_2 \in S$ for which row(s_1) = row(s_2), and thus our table is always consistent.) Then, from (S, E, T), the DFA $M = M(S, E, T)$ is constructed. Unlike in previous work, our algorithm now "experiments" with M to see if a counterexample can be found. More specifically, let τ contain the set of all strings from Σ^* of length at most $(c-2)\log_k n$. Our algorithm searches for a counterexample to M by simulating the execution of each string from τ starting at each state in M with the hope of reaching two different states of M. When no counterexample is found by this procedure then an equivalence query is made. We then use Schapire's procedure to find an experiment e from which we can construct a set of at least $(c-2)\log_k n$ experiments, each of which causes a new state to be added to S in order to re-establish closure. We now present an algorithm that assumes the learner knows n and then describe how to handle the case in which n is not known.

Theorem 21 *Let U be the unknown regular language over Σ that can be represented by an n-state*

New-Learn-DFA(n, Σ)

Initialize S and E to $\{\lambda\}$.
Construct the initial observation table (S, E, T)
Repeat
    While (S, E, T) is not closed
        Find $s_0 \in S, a \in \Sigma$ such that $\text{row}(s_0 \cdot a) \neq \text{row}(s)$ for all $s \in S$
        Add $s_0 \cdot a$ to S
        Extend T to $(S \cup S \cdot \Sigma) \cdot E$ using membership queries
    Let $M = M(S, E, T)$

    If there exists some $s \in S, t \in \tau$, and $e \in E$ such that $\gamma_M(s \cdot t \cdot e) \neq \text{MQ}(s \cdot t \cdot e)$
        Let $x \leftarrow s \cdot t \cdot e$
    Else if EQUIV(M) returns "no"
        Let x be the counterexample returned
    Else exit the loop

    Apply Schapire's procedure to x (making $\leq \lg |x|$ MQs) to find experiment e where
      for $a_0 \in \Sigma$ and $s_0, s_1 \in S$ with $\text{row}(s_1) = \text{row}(s_0 \cdot a_0)$, but $\gamma(s_1 \cdot e) \neq \gamma(s_0 \cdot a_0 \cdot e)$
    For each loop that exists when simulating M on e from state $s_0 \cdot a_0$
      Let $x \leftarrow$ string obtained by removing the loop from e
        If $\gamma(s_1 \cdot x) \neq \gamma(s_0 \cdot a_0 \cdot x)$ then $e \leftarrow x$

    Let the resulting string be represented by $a_1 \cdots a_r$ for $a_i \in \Sigma$
    If $Q_M(s_0 \cdot a_0 \cdot a_1 \cdots a_i) \neq Q_M(s_1 \cdot a_1 \cdots a_i)$ for $1 \leq i \leq r-1$
        Let ℓ be the smallest i for which this occurs
        Find $e' \in E$ such that $\gamma(s_0 \cdot a_0 \cdot a_1 \cdots a_\ell \cdot e') \neq \gamma(s_1 \cdot a_1 \cdots a_\ell \cdot e')$
          (So e' distinguishes state $s_0 \cdot a_0 \cdot a_1 \cdots a_\ell$ from state $s_1 \cdot a_1 \cdots a_\ell$)
    Else Let $\ell \leftarrow r$
        Let $e' \leftarrow \lambda$

    For $1 \leq i \leq \ell$
        Add experiment $a_i \cdots a_\ell \cdot e'$ to E
Until answer is "yes" from equivalence query.

Figure 6: An algorithm to learn a DFA with at most $(n-1)/(c \log_k n)$ equivalence queries when n is known a priori.

DFA, let $k = |\Sigma|$, *and let* $c \geq 2 + \frac{\lg k}{\lg n}$ *be some constant. Then*

$$\mathcal{E}\left(DFA_{n,\Sigma}, \frac{k}{k-1}(n^c + n^2) + n \lg n\right) \leq \left\lceil \frac{n-1}{(c-2)\log_k n} \right\rceil.$$

Proof: Let τ contain the set of all strings from Σ^* of length at most $(c-2)\log_k n$. For any $x \in \Sigma^*$, we let $Q_M(x)$ denote the state reached in DFA M when executing x from the initial state of M. Our algorithm is shown in Figure 6. Observe that, as in Schapire's algorithm, the observation table is always consistent since a new state is added to S only when there is evidence (namely, an experiment in E) that it is distinct from all other states in S. Furthermore, observe that S is prefix closed and E is suffix closed. We now restate the following theorem from Angluin [Ang87b].

Theorem 22 *If (S,E,T) is a closed, consistent observation table, then the acceptor M(S,E,T) is consistent with the finite function T. Any other acceptor consistent with T but inequivalent to M(S,E,T) must have more states.*

Thus it follows that once $|S| = n$, our algorithm will halt outputting an n-state DFA recognizing U. Below we argue that whenever the learner receives a counterexample from the equivalence oracle, at least $(c-2)\log_k n$ new (and distinct) states are added to S in the process of re-establishing closure. Thus, since initially $|S| = 1$, after at most $\left\lceil \frac{n-1}{(c-2)\log_k n} \right\rceil$ counterexamples are received $M(S,E,T)$ will exactly identify the target.

We now argue by contradiction that, whenever a counterexample provided by the equivalence oracle is processed, $\ell \geq (c-2)\log_k n$. Suppose that for $\sigma = a_1 \cdots a_\ell$, we have that $|\sigma| \leq (c-2)\log_k n - 1$. By the definition of ℓ, we have that $Q_M(s_0 \cdot a_0 \cdot \sigma) \neq Q_M(s_1 \cdot \sigma)$. And by the construction of the observation table, there must be an experiment $e' \in E$ for which $\gamma(s_0 \cdot a_0 \cdot \sigma \cdot e') \neq \gamma(s_1 \cdot \sigma \cdot e')$. Since the state $s_0 \cdot a_0 \cdot \sigma$ (respectively, $s_1 \cdot \sigma$) is reachable from s_0 (respectively, s_1) with at most $(c-2)\log_k n$ steps, during the simulation the learner would have considered both $s \cdot \sigma \cdot e'$ and $s' \cdot a_0 \cdot \sigma \cdot e'$, thus finding its own counterexample. This contradicts the fact that an equivalence query was made, and thus $\ell \geq (c-2)\log_k n$.

Next we argue inductively that at least ℓ new (and thus distinct) states are added to S to re-establish closure. Let $s_i = Q_M(s_1 \cdot a_1 \cdots a_{i-1})$ and $s'_i = Q_M(s_0 \cdot a_0 \cdot a_1 \cdots a_{i-1})$ for $1 \leq i \leq \ell$. Also, we use σ_i to denote the experiment $a_i \cdots a_\ell \cdot e'$. We use (S_0, E_0, T) to denote the observation table used to generate the automaton M used for the equivalence query, and (S_i, E_i, T) to denote the observation table where $E_i = E_0 \cup \{\sigma_1 \cup \sigma_2 \cup \cdots \cup \sigma_i\}$. We shall use row_i when referring to observation table (S_i, E_i, T). Since S is prefix-closed, observe that if $Q_M(s_{i_1}) = Q_M(s_{i_2}) = \cdots = Q_M(s_{i_t})$ for $i_1 < i_2 < \cdots < i_t$, then $s_{i_1} \in S_0$. For ease of exposition, whenever we say that $Q_M(s_i) \neq Q_M(s_j)$ for $1 \leq j \leq \ell$, we implicitly mean that $i \neq j$.

We use the following observations:

1. For $1 \leq i \leq \ell$, $Q_M(s_i) = Q_M(s'_i)$.

2. If $Q_M(s_i) \neq Q_M(s_j)$ then there exists an experiment $e \in E_0$, such that $\gamma(s_i \cdot e) \neq \gamma(s_j \cdot e)$.

3. For $1 \leq i \leq \ell - 1$, $\gamma(s'_i \cdot a_i \cdot \sigma_{i+1}) \neq \gamma(s_{i+1} \cdot \sigma_{i+1})$.

4. For $1 \leq i \leq \ell$, $s'_i \notin S_0$.

The first observation follows directly from the selection of ℓ. The second observation follows from the fact that if $Q_M(s_i) \neq Q_M(s_j)$ then $\text{row}_0(s_i) \neq \text{row}_0(s_j)$. The third observation follows from the fact that $\gamma(s'_i \cdot a_i \cdot \sigma_{i+1}) = \gamma(s_0 \cdot a_0 \cdot a_1 \cdots a_\ell \cdot \sigma) \neq \gamma(s_1 \cdot a_1 \cdots a_\ell \cdot \sigma) = \gamma(s_{i+1} \cdot \sigma_{i+1})$ for $1 \leq i \leq \ell - 1$. Finally, the fourth observation follows from the fact that S_0 is prefix-closed and $s_0 \cdot a_0 = s'_1 \notin S_0$.

We prove the following inductively for $1 \leq i \leq \ell$:

1. For $1 \leq j \leq i$, $|S_i - S_0| \geq i$.

2. There exists an $\varphi' \in S_i$ such that $Q_M(\varphi') = Q_M(s'_i)$ and $\gamma(s'_i \cdot \sigma_i) = \gamma(\varphi' \cdot \sigma_i)$.

3. Likewise, there exists an $\varphi \in S_i$ such that $Q_M(\varphi) = Q_M(s_i)$ and $\gamma(s_i \cdot \sigma_i) = \gamma(\varphi \cdot \sigma_i)$.

For the base case, observe that in re-establishing closure the experiment σ_1 will cause $s'_1 = s_0 \cdot a_0$ to be added to S yielding the observation table, (S_1, E_1, T). (The second two parts of the inductive hypothesis trivially hold since s_1 and s'_1 are in S_1.) We now assume that the inductive hypothesis holds for $1 \leq i \leq \ell - 1$, and prove that it holds for $i + 1$. We consider the following two cases:

Case 1: $Q_M(s_{i+1}) \neq Q_M(s_j)$ for $j < i + 1$. Let $\varphi' \in S_i$ be such that $Q_M(\varphi') = Q_M(s'_i)$ and $\gamma(s'_i \cdot \sigma_i) = \gamma(\varphi' \cdot \sigma_i)$. Since $Q_M(s'_i \cdot a_i) = s_{i+1}$, it follows from the inductive hypothesis that $Q_M(\varphi' \cdot a_i) = s_{i+1}$. From Observation 2, it immediately follows that since $Q_M(\varphi' \cdot a_i) = Q_M(s_{i+1})$, $\text{row}_i(\varphi' \cdot a_i) \neq \text{row}_i(s)$ for any $s \in S_i$ except s_{i+1}. Thus we need just show that $\text{row}_{i+1}(\varphi' \cdot a_i) \neq \text{row}_{i+1}(s_{i+1})$. This follows immediately from the fact that $\gamma(\varphi' \cdot a_i \cdot \sigma_{i+1}) \neq \gamma(s_{i+1} \cdot \sigma_{i+1})$. Thus the new state $\varphi' \cdot a_i$ is added to S_i yielding S_{i+1}. Since $s_{i+1} \in S_{i+1}$, the last part of the inductive hypothesis hold.

Case 2: $Q_M(s_{i+1}) = s_j$. Let $\varphi \in S_i$ be such that $Q_M(\varphi) = Q_M(s_j)$ and $\gamma(s_j \cdot \sigma_i) = \gamma(\varphi \cdot \sigma_i)$, and let $\varphi' \in S_i$ be such that $Q_M(\varphi') = Q_M(s'_j)$ and $\gamma(s'_j \cdot \sigma_i) = \gamma(\varphi' \cdot \sigma_i)$. As in Case 1, it is easily shown that $\text{row}_i(\varphi' \cdot a_i) \neq \text{row}_i(s)$ for any $s \in S_i$ such that $Q_M(s \cdot a_i) \neq s_j$. Thus we need just show that $\text{row}_{i+1}(s_{i+1})$ (or $\text{row}_{i+1}(s'_{i+1})$) is distinct from both $\text{row}_{i+1}(\varphi)$ and $\text{row}_{i+1}(\varphi')$. Recall that $\gamma(s_{i+1} \cdot \sigma_{i+1}) \neq \gamma(s'_{i+1} \cdot \sigma_{i+1})$. Since the loop $a_j \cdots a_i$ could not be removed, it follows that $\gamma(s_j \cdot \sigma_{i+1}) = \gamma(s'_j \cdot \sigma_{i+1})$. We consider the following two subcases.

Case: 2a: $\gamma(\varphi' \cdot \sigma_{i+1}) = \gamma(\varphi \cdot \sigma_{i+1}) = \gamma(s_{i+1} \cdot \sigma_{i+1})$. Thus, it follows that σ_{i+1} distinguishes s'_{i+1} from both φ and φ'. That is, $\gamma(s'_{i+1} \cdot \sigma_{i+1}) \neq \gamma(\varphi \cdot \sigma_{i+1})$ and $\gamma(s'_{i+1} \cdot \sigma_{i+1}) \neq \gamma(\varphi' \cdot \sigma_{i+1})$. Thus it follows that $\varphi' \cdot a_i$ will be placed in S_{i+1}. Furthermore, since $\gamma(s_{i+1} \cdot \sigma_{i+1}) = \gamma(\varphi \cdot \sigma_{i+1})$, there exists a $\varphi'' \in S_{i+1}$ such that $\gamma(s_{i+1} \cdot \sigma_{i+1}) = \gamma(\varphi'' \cdot \sigma_{i+1})$.

Case: 2b: $\gamma(\varphi' \cdot \sigma_{i+1}) = \gamma(\varphi \cdot \sigma_{i+1}) = \gamma(s'_{i+1} \cdot \sigma_{i+1})$. Thus, it follows that σ_{i+1} distinguishes s_{i+1} from both φ and φ'. The rest of this argument is analogous to Case 2a.

Thus, once S_ℓ has been reached, we have that $|S_\ell - S_0| \geq \ell$ proving that after obtaining closure, $|S|$ has increased by at least ℓ from the point at which the equivalence query was made.

To obtain the desired bound on the number of membership queries observe that $|S| \leq n$ and $|E| \leq n$. Furthermore, since no entries are ever removed from the observation table, the total number of membership queries used to fill in the observation table and do the needed exploration is at most $n^2 + n^2|\tau|$ where $|\tau| = \sum_{i=1}^{(c-2)\lg_k n} k^i = \frac{k}{k-1}(n^{c-2} - 1)$. In addition, at most $\lg n$ membership queries are used by Schapire's procedure after receiving each counterexample, and at most n membership queries are used at each iteration when trying to remove loops. The stated bound follows from algebra and the observation that $1/(c-2) \leq \lg n / \lg k$. □

We now show how the algorithm given in Figure 6 can be modified to work when the number of states, n, in the target DFA is not provided to the learner.

Theorem 23 *Let U be the unknown regular language over Σ that can be represented by an n state DFA, let $k = |\Sigma|$, and let $c \geq 2 + \frac{\log k}{\log n}$ be some constant. Then*

$$\mathcal{E}(DFA_\Sigma, O(n^c)) \leq \sqrt{n}\left(1 - \frac{2\lg k}{\lg n}\right) + \left\lceil \frac{2(n-1)}{(c-2)\log_k n} \right\rceil.$$

Proof: The learner uses the algorithm of Figure 6 except that instead of having τ contain all strings of length $(c-2)\log_k n$, we replace it by a set τ' that contains all strings of length $(c-2)\log_k w$ where w is the maximum of k and the number of states in the previously conjectured DFA. Then just replace τ by τ' in **New-Learn-DFA**.

Now clearly each of the first $\sqrt{n}$ equivalence queries enables the learner to find at least $c-2$ new states of the target since every string in τ' is of length at least $c-2$. For the remaining portion of the algorithm $w \geq \sqrt{n}$ thus each additional equivalence query enables the learner to find at least $\frac{(c-2)}{2}\log_k n$ new states of the target. Thus the number of equivalence queries used by the learner is at most

$$\sqrt{n} + \left\lceil \frac{2(n - 1 - (c-2)\sqrt{n})}{(c-2)\log_k n} \right\rceil \leq \sqrt{n}\left(1 - \frac{2\lg k}{\lg n}\right) + \left\lceil \frac{2(n-1)}{(c-2)\log_k n} \right\rceil.$$

□

11 Concluding Remarks

This work has established some sharp bounds for a variety of the most basic classes for which exact identification algorithms are known. It is perhaps surprising that we had such success in proving matching lower and upper bounds, particularly since the lower bounds hold under the most

favorable conditions for learning (arbitrary hypotheses and superpolynomial time), while the upper bounds, in general, hold under the most restrictive (hypotheses must be in the class to be learned, polynomial time is required). The one exception to these sharp bounds is for Horn sentences where it remains an open problem whether our tight upper bound given unlimited computation can be achieved by a polynomial time algorithm.

Other open problems include applying these techniques to find tight bounds for other classes for which polynomial time exact identification algorithms are known, such as decision trees [Bsh93], and read-twice DNF formulas [AP91, Han91, PR93].

Another interesting problem is how the use of membership queries can reduce the number of equivalence queries needed to learn classes that can in fact be learned with equivalence queries alone, such as k-DNF. For k-DNF, Littlestone's algorithm uses $O(k\ell \log n)$ equivalence queries where ℓ is the number of terms [Lit88]. A lower bound is $n \log n$. The gap is still open, even with using membership queries (though if ℓ is known, our generalized halving algorithm uses $O\left(\frac{k\ell \log n}{d(\log k\ell + \log\log n)}\right)$ queries).

Acknowledgments

We thank Tino Tamon for many useful discussions. Nader Bshouty and Sleiman Matar are supported in part by the NSERC of Canada. Sally Goldman is supported in part NSF grant CCR-91110108. This work was begun while Tom Hancock was at Harvard University, supported by ONR grant N00014-85-K-0445 and NSF grant NSF-CCR-89-02500.

References

[AFP90] Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of horn clauses. In *31st Annual Symposium on Foundations of Computer Science*, pages 186–192, October 1990.

[AFP92] Dana Angluin, Michael Frazier, and Leonard Pitt. Learning conjunctions of horn clauses. *Machine Learning*, 9:147–164, 1992. Special Issue for COLT 90.

[AHK89] Dana Angluin, Lisa Hellerstein, and Marek Karpinski. Learning read-once formulas with queries. Technical Report UCB/CSD 89/528, University of California Berkeley Computer Science Division, 1989.

[Ang87a] Dana Angluin. Learning k-term DNF formulas using queries and counterexamples. Technical Report YALEU/DCS/RR-559, Yale University, August 1987.

[Ang87b] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[Ang88] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[Ang90] Dana Angluin. Negative results for equivalence queries. *Machine Learning*, 5:121–150, 1990.

[AP91] Howard Aizenstein and Leonard Pitt. Exact learning of read-twice DNF formulas. In *32nd Annual Symposium on Foundations of Computer Science*, pages 170–179, October 1991.

[AP92] Howard Aizenstein and Leonard Pitt. Exact learning of read-k disjoint DNF and not-so-disjoint DNF. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 71–76, July 1992.

[BC92] Nader H. Bshouty and Richard Cleve. On the exact learning of formulas in parallel. In *33rd Annual Symposium on Foundations of Computer Science*, pages 1–15, October 1992.

[BF72] IAn Barzdin and Rūsiņš Freivald. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

[BHH92a] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning arithmetic read-once formulas. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 370–381, May 1992.

[BHH92b] Nader H. Bshouty, Thomas R. Hancock, and Lisa Hellerstein. Learning Boolean read-once formulas with arbitrary symmetric and constant fan-in gates. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 1–15, August 1992. To appear, *Journal of Computer and Systems Sciences.*

[BHHK91] Nader H. Bshouty, Thomas R. Hancock, Lisa Hellerstein, and Marek Karpinski. Learning Boolean read-once formulas with arbitrary symmetric and constant fan-in gates. Technical Report TR-92-020, International Computer Science Institute, 1991. To appear, *Computational Complexity.*

[BM76] John A. Bondy and U. S. R. Murty. *Graph theory with applications.* Macmillan, London, 1977, c1976.

[BR92] Avrim Blum and Steven Rudich. Fast learning of k-term DNF formulas with queries. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 382–389, May 1992.

[Bsh93] Nader H. Bshouty. Exact learning. Unpublished manuscript, 1993.

[Han91] Thomas R. Hancock. Learning 2μ DNF formulas and $k\mu$ decision trees. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 199–209, August 1991.

[HKLW88] David Haussler, Michael Kearns, Nick Littlestone, and Manfred K. Warmuth. Equivalence of models for polynomial learnability. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 42–55. Morgan Kaufmann, August 1988.

[Lit88] Nick Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[MT92] Wolfgang Maass and György Turán. Lower bound methods and separation results for on-line learning models. *Machine Learning*, 9:107–145, 1992.

[NN90] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 213–223, May 1990.

[PR93] Krishnan Pillapakkamnatt and Vijay Raghavan. Read-Twice DNF Formulas are Properly Learnable (Revised). Technical Report TR-CS-93-59, Department of Computer Science, Vanderbilt University.

[RS89] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 411–420, May 1989.

[RS90] Vijay Ragavan and Stephen R. Schach. Learning switch configurations. In *Proceedings of the 1990 Workshop on Computational Learning Theory*, pages 38–51. Morgan Kaufmann, August 1990.

[Sc91] Robert E. Schapire. *The Design and Analysis of Efficient Learning Algorithms.* MIT Press, Cambridge, Massachusetts, 1991.

[Val84] Leslie Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.