Learning Functions Represented as Multiplicity Automata

AMOS BEIMEL

Harvard University, Cambridge, Massachusetts

FRANCESCO BERGADANO

Università di Torino, Torino, Italy

NADER H. BSHOUTY

University of Calgary, Calgary, Alberta, Canada

EYAL KUSHILEVITZ

Technion, Haifa, Israel

AND

STEFANO VARRICCHIO

Universitá di L'Aquila, L'Aquila, Italy

Abstract. We study the learnability of multiplicity automata in Angluin's *exact learning model*, and we investigate its applications. Our starting point is a known theorem from automata theory relating the

This paper contains results from three conference papers. It is mostly based on the FOCS '96 paper by the authors. The rest of the results are from the STOC '96 paper of Bergadano, Catalano, and Varricchio and the EuroCOLT '97 paper of Beimel and Kushilevitz.

Part of this research was done while A. Beimel was a Ph.D student at the Technion.

The research of F. Bergadano was funded in part by the European Community under grant 20237 (ILP2).

The research of E. Kushilevitz was supported by Technion V.P.R. Fund 120-872 and by the Japan-Technion Society Research Fund.

Authors' addresses: A. Beimel, Division of Engineering and Applied Sciences, Harvard University, 40 Oxford Street, Cambridge, MA 02139, e-mail: beimel@deas.harvard.edu, http://www.deas.harvard.edu/~beimel; F. Bergadano, Università di Torino, e-mail: bergulan@di.unito.it; N. H. Bshouty, Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, e-mail: bshouty@cpcs.ucalgary.ca, http://www.cpsc.ucalgary.ca/~bshouty; E. Kushilevitz, Department of Computer Science, Technion, Haifa 32000, Israel, e-mail: eyalk@cs.technion.ac.il, http://www.cs.technion.ac.il/~eyalk; S. Varricchio, Università di L'Aquila, e-mail: varricch@univaq.it.

Journal of the ACM, Vol. 47, No. 5, May 2000, pp. 506–530.

number of states in a minimal multiplicity automaton for a function to the rank of its Hankel matrix. With this theorem in hand, we present a new simple algorithm for learning multiplicity automata with improved time and query complexity, and we prove the learnability of various concept classes. These include (among others):

—The class of disjoint DNF, and more generally satisfy-$O(1)$ DNF.

—The class of polynomials over finite fields.

—The class of bounded-degree polynomials over infinite fields.

—The class of XOR of terms.

—Certain classes of boxes in high dimensions.

In addition, we obtain the best query complexity for several classes known to be learnable by other methods such as decision trees and polynomials over GF(2).

While multiplicity automata are shown to be useful to prove the learnability of some subclasses of DNF formulae and various other classes, we study the limitations of this method. We prove that this method cannot be used to resolve the learnability of some other open problems such as the learnability of general DNF formulas or even k-term DNF for $k = \omega(\log n)$ or satisfy-s DNF formulas for $s = \omega(1)$. These results are proven by exhibiting functions in the above classes that require multiplicity automata with super-polynomial number of states.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation; F.2 [**Analysis of Algorithms and Problem Complexity**]; I.2.6 [**Artificial Intelligence**]: Learning

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Computational learning, DNF, learning disjoint, DNF, learning polynomials, multiplicity automata

1. *Introduction*

A central task of learning theory is the classification of classes of functions into those which are (efficiently) learnable and those which are not (efficiently) learnable. This task, in spite of enormous efforts, is still far from being accomplished under any of the major learning models. This work presents techniques and results which are targeted toward this goal, with respect to the important *exact learning* model.

The *exact learning* model was introduced by Angluin [1988] and since then attracted a lot of attention. It represents situations in which a learner who tries to learn some target function f is allowed to actively collect information regarding f; this is in contrast to other major learning models, such as the *Probably Approximately Correct (PAC)* model of Valiant [1984], in which the learner tries to "learn" f by passively observing examples (of the form x, $f(x)$). The way by which the learning algorithm collects information about the target function f, in the exact learning model, is by asking *queries*: the learning algorithm may ask for the value $f(x)$ on points x of its choice (this is called *membership query*) or it may suggest a hypothesis h to which it gets a counterexample x if such exists (i.e., x such that $f(x) \neq h(x)$). The second type of queries is called *equivalence query*. (For a formal definition of the model, see Section 2.2.) One of the basic observations regarding this model is that every equivalence query can be easily simulated by a sample of random examples. Therefore, learnability in the exact learning model also implies learnability in the PAC model with membership queries [Valiant 1984; Angluin 1988]. Attempts to prove learnability of various classes in the exact learning model were made in several directions. In particular, the learnability of *DNF formulas* and various kinds of *automata* attracted a lot of attention.

1.1 DNF FORMULAS. While general formulas are not learnable in this model [Kharitonov 1993], as well as in the other major models of learning, researchers concentrated on the question of learning the class of DNF (Disjunctive Normal Form) formulas. The learnability of this class is still an open problem in most of the major models of learning.[1] The attraction of this class stems from several facts: On one hand, it seems like a simple, natural class that is only "slightly above" our current state of knowledge, and on the other hand, it appears that people like it for representing knowledge [Valiant 1985]. Much work was therefore devoted to learning subclasses of DNF formulas.[2] These subclasses are obtained by restricting the DNF formulas in various ways; for example, by limiting the number of terms in the target formula or by limiting the number of appearances of each variable. The motivation for studying these subclasses is that such results may shed some light on the general question of learning the whole class of DNF formulas, and that it might be that in practice functions of interest belong to these subclasses. Moreover, it might as well be the case that the whole class of DNF formulae is not at all learnable.

Another class whose learnability is of a wide interest in practice is that of (Boolean) *decision trees* [Breiman et al. 1984; Quinlan 1986; 1993]. This class was shown to be learnable in the exact learning model in Bshouty [1995a]. It is not hard to see that every decision tree can be transformed into a DNF formulas of essentially the same size. Hence, this class can also be considered as a subclass of DNF formulas. Looking closely at the DNF formulas obtained by this transformation, we see that any assignment x satisfies at most one term of the formula. A DNF formula with this property is called *disjoint* DNF. This class of formulas was the subject of previous research but, once again, only subclasses of disjoint DNF were known to be learnable prior to our work [Aizenstein and Pitt 1992; Blum et al. 1994].

1.2. AUTOMATA. One of the first classes shown to be learnable in the exact learning model is that of deterministic automata [Angluin 1987]. This class is interesting both from the practical point of view, since finite state machines model many behaviors that we wish to learn in practice (see Trakhtenbrot and Barzdin [1973] and Lang [1992], and references therein), and since this particular class in known to be hard to learn from examples only, under common cryptographic assumptions Kearns and Valiant [1994]. This result of Angluin was extended by Bergadano and Varricchio [1996a] and Ohnishi et al. [1994] where the class of *multiplicity automata* was shown to be learnable in the exact learning model. Multiplicity automata are essentially nondeterministic automata with weights from a field $\mathcal{K}$ on the edges. Such an automaton computes a function f as follows: For every path in the automaton, assign a weight which is the product of the weights on the edges of this path. The value $f(x)$ computed by the automaton is essentially the sum of the weights of the paths consistent with the input string

[1] With the exception of Jackson [1997], who showed the learnability of this class using membership queries with respect to the uniform distribution.

[2] See, for example, Aizenstein and Pitt [1991; 1992], Angluin [1987a], Angluin et al. [1993], Blum et al. [1994], Blum and Rudich [1995], Bshouty [1995a; 1997], Hancock [1991], and Kushilevitz [1997].

x (this sum is a value in $\mathcal{K}$).[3] Multiplicity automata were first introduced by Shützenberger [1961], and have been used as a mathematical tool for modeling probabilistic automata and ambiguity in nondeterministic automata. They are also widely used in the theory of rational series in noncommuting variables. Multiplicity automata are a generalization of deterministic automata, and the algorithms that learn this class [Bergadano and Varricchio 1996a; Ohnishi et al. 1994] are generalizations of Angluin's algorithm for deterministic automata [Angluin 1987].

1.3. OUR RESULTS. In this work, we find connections between the learnability questions of some of the above-mentioned classes and other classes of interest. More precisely, we show that the learnability of multiplicity automata implies the learnability of many other important classes of functions. In Kushilevitz [1997], it is shown how the learnability of *deterministic* automata can be used to learn certain classes of functions. These classes however are much more restricted and hence it yields much weaker results. Below, we give a detailed account of our results.

1.3.1. *Results on DNF Formulas.* First, it is shown that the learnability of multiplicity automata implies the learnability of the class of disjoint DNF and more generally of satisfy-s DNF formulas, for $s = O(1)$ (i.e., DNF formulas in which each assignment satisfies at most s terms). As mentioned, this class includes as a special case the class of decision trees. These results improve over previous results of Bshouty [1995a], Aizenstein and Pitt [1992], and Blum et al. [1994].

1.3.2. *Results on Geometric Boxes.* An important generalization of DNF formulas is that of boxes over a discrete domain of points (i.e., $\{0, \ldots, \ell - 1\}^n$). Such boxes were considered in many works.[4] We prove the learnability of any union of $O(\log n)$ boxes in time poly(n, ℓ), and the learnability of any union of t disjoint boxes (and more generality, any t boxes such that each point is contained in at most $s = O(1)$ of them) in time poly(n, t, ℓ).[5] The special case of these results where $\ell = 2$ implies the learnability of the corresponding classes of DNF formulas.

1.3.3. *Results on Polynomials.* Multivariate polynomials can be viewed as an algebraic analogue of DNF formulas. However, they are also of independent interest. In particular, the question of learning polynomials from membership queries only is just the fundamental *interpolation* problem, which was studied in numerous papers (see, e.g., Zippel [1993]). Since learning polynomials over small fields with membership queries only is not possible, it raises the question whether with the help of equivalence queries this becomes possible. Before the current work, it was known how to learn the class of multi-linear polynomials and polynomials over GF(2) [Schapire and Sellie 1996]. We further show the learnability of the class of XOR of terms, which is an open problem in Schapire

[3] These automata are known in the literature under various names. In this paper, we refer to them as *multiplicity automata*. The functions computed by these automata are usually referred to as *recognizable series*.

[4] See, for example, Maass and Turán [1989; 1994], Chen and Maass [1992], Auer [1993], Goldberg et al. [1994], Jackson [1997], and Maass and Warmuth [1998].

[5] In Beimel and Kushilevitz [1998], using additional machinery, the dependency on ℓ was improved.

and Sellie [1996], the class of polynomials over finite fields, which is an open problem in Schapire and Sellie [1996] and Bshouty [1995b], and the class of bounded-degree polynomials over infinite fields (as well as other classes of functions over finite and infinite fields).

1.3.4. *Techniques*. We use an algebraic approach for learning multiplicity automata, similar to Ohnishi et al. [1994]. This approach is based on a fundamental theorem in the theory of multiplicity automata. The theorem relates the size of a smallest automaton for a function f to the rank (over $\mathcal{K}$) of the so-called Hankel matrix of f [Carlyle and Paz 1971; Fliess 1974] (see also Eilenberg [1974] and Berstel and Reutenauer [1988] for background on multiplicity automata). Using this theorem, and ideas from the algorithm of Rivest and Schapire [1993] (for learning deterministic automata), we develop a new algorithm for learning multiplicity automata which is more efficient than the algorithms of Bergadano and Varricchio [1996a] and Ohnishi et al. [1994]. In particular, we give a more refined analysis for the complexity of our algorithm when learning functions f with finite domain. A different algorithm with similar complexity to ours was found by Bshouty et al. [1998].[6]

1.3.5. *Negative Results*. While multiplicity automata are proved to be useful to solve many open problems regarding the learnability of subclasses of DNF and other classes of polynomials and decision trees, we study the limitations of this method. We prove that this method cannot be used to resolve the learnability of some other open problems such as the learnability of general DNF formulas or even k-term DNF for $k = \omega(\log n)$ (a function is in the class k-term DNF if it can be represented by a DNF formula with at most k terms) or satisfy-s DNF formulas for $s = \omega(1)$ (these results are tight in the sense that $O(\log n)$-term DNF formulas and satisfy-$O(1)$ DNF formulas are learnable using multiplicity automata). These negative results are proven by exhibiting functions in the above classes that require multiplicity automata with super-polynomial number of states. For proving these results, we use, again, the relation between multiplicity automata and Hankel matrices.

1.4. ORGANIZATION. In Section 2, we present some background on multiplicity automata, as well as the definition of the learning model. In Section 3, we present a learning algorithm for multiplicity automata. In Section 4, we present applications of the algorithm for learning various classes of functions. Finally, in Section 5, we study the limitations of this method.

2. *Background*

2.1. MULTIPLICITY AUTOMATA. In this section, we present some definitions and a basic result concerning multiplicity automata. Let $\mathcal{K}$ be a field, Σ be an alphabet, ϵ be the empty string, and $f: \Sigma^* \rightarrow \mathcal{K}$ be a function. Associate with f an infinite matrix F each of its rows is indexed by a string $x \in \Sigma^*$ and each of its columns is indexed by a string $y \in \Sigma^*$. The (x, y) entry of F contains the value $f(x \circ y)$, where $\circ$ denotes concatenation. (In the automata literature, such a function f is often referred to as a *formal series* and F as its *Hankel Matrix*.) We

[6] In fact, Bshouty et al. [1998] show that the algorithm can be generalized to $\mathcal{K}$, which is not necessarily a field but rather a certain type of ring.

use F_x to denote the xth row of F. The (x, y) entry of F may be therefore denoted as $F_x(y)$ and as $F_{x,y}$. The same notation is adapted to other matrices used in the sequel.

Next we define the automaton representation (over the field $\mathcal{K}$) of functions. An automaton A of size r consists of $|\Sigma|$ matrices $\{\mu_\sigma: \sigma \in \Sigma\}$ each of which is an $r \times r$ matrix of elements from $\mathcal{K}$ and an r-tuple $\vec{\gamma} = (\gamma_1, \ldots, \gamma_r) \in \mathcal{K}^r$. The automaton A defines a function f_A: $\Sigma^* \rightarrow \mathcal{K}$ as follows: First, define a mapping μ, which associates with every string in Σ^* an $r \times r$ matrix over $\mathcal{K}$, by $\mu(\epsilon) \triangleq \mathrm{ID}$, where ID denotes the *identity matrix*,[7] and for a string $w = \sigma_1\sigma_2 \cdots \sigma_n$, let $\mu(w) \triangleq \mu_{\sigma_1} \cdot \mu_{\sigma_2} \cdots \mu_{\sigma_n}$ (a simple but useful property of μ is that $\mu(x \circ y) = \mu(x) \cdot \mu(y)$). Now, $f_A\ (w) \triangleq [\mu(w)]_1 \cdot \vec{\gamma}$ (where $[\mu(w)]_i$ denotes the ith row of the matrix $\mu(w)$). In words, A is an automaton with r states where q_1 is the initial state and the transition from state q_i to state q_j with letter σ has weight $[\mu_\sigma]_{i,j}$. The first row of the matrices μ_σ corresponds to the initial state of the automaton (this is the intuition why the definition of f_A uses only the first row of $\mu(w)$). The weight of a path whose last state is q_ℓ is the product of weights along the path multiplied by γ_ℓ, and the function computed on a string w is just the sum of weights over all paths corresponding to w.

Example 2.1 [*Berstel and Reutenauer* 1988]. Let $\Sigma = \{a, b\}$ and define the function $\#_a$: $\Sigma \rightarrow \mathcal{Q}$, where $\#_a(w)$ is the number of times that the letter a appears in w. The function $\#_a$ has a multiplicity automaton (over $\mathcal{Q}$) of size 2, where $\vec{\gamma} = (0, 1)$ and

$$\mu_a = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \qquad \mu_b = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Let α be the number of times that a appears in w. The mapping $\mu(w)$ is therefore

$$\mu(w) = \begin{pmatrix} 1 & \alpha \\ 0 & 1 \end{pmatrix}$$

(this can be easily proved by induction). Thus, the function computed by the automaton is $[\mu(w)]_1 \cdot \vec{\gamma} = (1, \alpha) \cdot (0, 1) = \alpha$ as promised.

Example 2.2. *Probabilistic automata* are a simple example of multiplicity automata over the reals. In this case the sum of the weights in each row of μ_σ is 1, the entry $[\mu_\sigma]_{i,j}$ is the probability to move from q_i to q_j with an input letter σ, and $[\mu(w)]_{i,j}$ is the probability to move from q_i to q_j with an input string w.

Example 2.3. Consider multiplicity automata over $GF(2)$. These automata can be viewed as nondeterministic automata in which the acceptance criterion is changed; a string is accepted by the multiplicity automaton if the number of accepting paths is odd and the function computed by the automaton is the characteristic function of the language accepted by the automaton.

The following theorem of Carlyle and Paz [1971] and Fliess [1974] is a fundamental theorem from the theory of formal series. It relates the size of the minimal automaton for f to the rank of F.

[7] That is, a matrix with 1's on the main diagonal and 0's elsewhere.

THEOREM 2.4 [CARLYLE AND PAZ 1971; FLEISS 1974]. *Let $f: \Sigma^* \rightarrow \mathcal{K}$ such that $f \not\equiv 0$ and let F be the corresponding Hankel matrix. Then, the size r of the smallest automaton A such that $f_A \equiv f$ satisfies $r = rank(F)$ (over the field $\mathcal{K}$).*

Although this theorem is very basic, we provide its proof here as it sheds light on the way the algorithm of Section 3 works.

Direction I. Given an automaton A for f of size r, we prove that $\text{rank}(F) \leq r$. Define two matrices: R whose rows are indexed by Σ^* and its columns are indexed by $1, \ldots, r$ and C whose columns are indexed by Σ^* and its rows are indexed by $1, \ldots, r$. The (x, i) entry of R contains the value $[\mu(x)]_{1,i}$ and the (i, y) entry of C contains the value $[\mu(y)]_i \cdot \vec{\gamma}$. We show that $F = R \cdot C$. This follows from the following sequence of simple equalities:

$$\begin{aligned} F_{x,y} &= f(x \circ y) = f_A(x \circ y) = [\mu(x \circ y)]_1 \cdot \vec{\gamma} \\ &= [\mu(x)\mu(y)]_1 \cdot \vec{\gamma} = \sum_{i=1}^{r} [\mu(x)]_{1,i} \cdot [\mu(y)]_i \cdot \vec{\gamma} \\ &= R_x \cdot C^y, \end{aligned}$$

where C^y denotes the yth column of C. Obviously, the rank of both R and C is bounded by r. By linear algebra, $\text{rank}(F)$ is at most $\min\{\text{rank}(R), \text{rank}(C)\}$ and therefore $\text{rank}(F) \leq r$, as needed.

Direction II. Given a function f such that the corresponding matrix F has rank $r > 0$, we show how to construct an automaton A of size r that computes this function. Let $F_{x_1}, F_{x_2}, \ldots, F_{x_r}$ be r independent rows of F (i.e., a basis) corresponding to strings $x_1 = \epsilon, x_2, \ldots, x_r$. (It holds that $F_\epsilon \neq 0$ since $f \not\equiv 0$, thus F_ϵ can always be extended to a basis of the row space of F.) To define A, we first define $\vec{\gamma} = (f(x_1), \ldots, f(x_r))$. Next, for every σ, define the ith row of the matrix μ_σ as the (unique) coefficients of the row $F_{x_i \circ \sigma}$ when expressed as a linear combination of $F_{x_1}, \ldots, F_{x_r}$. That is,

$$F_{x_i \circ \sigma} = \sum_{j=1}^{r} [\mu_\sigma]_{i,j} \cdot F_{x_j}. \tag{1}$$

We will prove, by induction on $|w|$ (the length of the string w), that $[\mu(w)]_i \cdot \vec{\gamma} = f(x_i \circ w)$ for all i. It follows that $f_A(w) = [\mu(w)]_1 \cdot \vec{\gamma} = f(x_1 \circ w) = f(w)$ (as we choose $x_1 = \epsilon$). The induction base is $|w| = 0$ (i.e., $w = \epsilon$). In this case, we have $\mu(\epsilon) = \text{ID}$ and hence $[\mu(\epsilon)]_i \cdot \vec{\gamma} = \gamma_i = f(x_i) = f(x_i \circ \epsilon)$, as needed. For the induction step, we observe, using Eq. (1), that

$$f(x_i \circ \sigma \circ w) = F_{x_i \circ \sigma}(w) = \sum_{j=1}^{r} [\mu_\sigma]_{i,j} \cdot F_{x_j}(w).$$

Since $F_{x_j}(w) = f(x_j \circ w)$, then by induction hypothesis this equals

$$\sum_{j=1}^{r} [\mu_\sigma]_{i,j} \cdot [\mu(w)]_j \cdot \vec{\gamma} = [\mu(\sigma) \cdot \mu(w)]_i \cdot \vec{\gamma} = [\mu(\sigma \circ w)]_i \cdot \vec{\gamma},$$

as needed.

2.2. The Learning Model. The learning model we use is the *exact learning* model [Angluin 1988]: Let f be a *target* function. A learning algorithm may propose, in each step, a hypothesis function h by making an *equivalence query* (EQ) to an oracle. If h is equivalent to f on all input assignments then the answer to the query is YES and the learning algorithm succeeds and halts. Otherwise, the answer to the equivalence query is NO and the algorithm receives a *counterexample*–an assignment z such that $f(z) \neq h(z)$. The learning algorithm may also query an oracle for the value of the function f on a particular assignment z by making a *membership query* (MQ) on z. The response to such a query is the value $f(z)$.[8] We say that the learner *learns* a class of functions $\mathcal{C}$, if, for every function $f \in \mathcal{C}$, the learner outputs a hypothesis h that is equivalent to f and does so in time polynomial in the "size" of a shortest representation of f and the length of the longest counterexample.

3. *The Algorithm*

In this section, we describe an exact learning algorithm for multiplicity automata. The "size" parameters in the case of multiplicity automata are the number of states in a minimal automaton for f, and the size of the alphabet. The algorithm will be efficient in these numbers and the length of the longest counterexample provided to it.

Let $f: \Sigma^* \to \mathcal{K}$ be the target function. All algebraic operations in the algorithm are done in the field $\mathcal{K}$.[9] The algorithm learns a function f using its Hankel matrix representation, F. The difficulty is that F is infinite (and is very large even when restricting the inputs to some length n). However, Theorem 2.4 (Direction II) implies that it is essentially sufficient to maintain $r = \text{rank}(F)$ linearly independent rows from F; in fact, $r \times r$ submatrix of F of full rank suffices. Therefore, the learning algorithm can be viewed as a search for appropriate r rows and r columns.

The algorithm works in iterations. At the beginning of the ℓth iteration, the algorithm holds a set of rows $X \subset \Sigma^*$ $(X = \{x_1, \ldots, x_\ell\})$ and a set of columns $Y \subset \Sigma^*$ $(Y = \{y_1, \ldots, y_\ell\})$. Let $\hat{F}_z$ denote the restriction of the row F_z to the ℓ coordinates in Y, that is, $\hat{F}_z \triangleq (F_z(y_1), \ldots, F_z(y_\ell))$. Note that given z and Y the vector $\hat{F}_z$ is computed using $|Y|$ membership queries. It will hold that $\hat{F}_{x_1}, \ldots, \hat{F}_{x_\ell}$ are ℓ linearly independent vectors. Using these vectors the algorithm constructs a hypothesis h, in a manner similar to the proof of Direction II of Theorem 2.4, and asks an equivalence query. A counterexample to h leads to adding a new element to each of X and Y in a way that preserves the above properties. This immediately implies that the number of iterations is

[8] If f is Boolean, this is the standard membership query.

[9] We assume that every arithmetic operation in the field takes one time unit.

bounded by r. We assume without loss of generality that $f(\epsilon) \neq 0$.[10] The algorithm works as follows:

(1) Initialize: $x_1 \leftarrow \epsilon$, $y_1 \leftarrow \epsilon$, $X \leftarrow \{x_1\}$, $Y \leftarrow \{y_1\}$, and $\ell \leftarrow 1$.

(2) Define a hypothesis h (following Direction II of Theorem 2.4):
   Let $\vec{\gamma} = (f(x_1), \ldots, f(x_\ell))$. For every σ, define a matrix $\hat{\mu}_\sigma$ by letting its i-th row be the coefficients of the vector $\hat{F}_{x_i \circ \sigma}(y)$ when expressed as a linear combination of the vectors $\hat{F}_{x_1}, \ldots, \hat{F}_{x_\ell}$ (such coefficients exist as $\hat{F}_{x_1}, \ldots, \hat{F}_{x_\ell}$ are ℓ independent ℓ-tuples).
   That is, $\hat{F}_{x_i \circ \sigma}(y) = \Sigma_{j=1}^{\ell} [\hat{\mu}_\sigma]_{i,j} \cdot \hat{F}_{x_j}$.
   For $w \in \Sigma^*$ define an $\ell \times \ell$ matrix $\hat{\mu}(w)$ as follows: Let $\hat{\mu}(\epsilon) = \mathrm{ID}$ and for a string $w = \sigma_1 \cdots \sigma_k$, let $\hat{\mu}(w) = \hat{\mu}_{\sigma_1} \cdot \hat{\mu}_{\sigma_2} \cdots \hat{\mu}_{\sigma_k}$. Finally, h is defined as $h(w) = [\hat{\mu}(w)]_1 \cdot \vec{\gamma}$.[11]

(3) Ask an equivalence query EQ(h).
   If the answer is YES halt with output h.
   Otherwise the answer is NO and z is a counterexample.
   Find (using MQs for f) a string $w \circ \sigma$ which is a prefix of z such that:
   (a) $\hat{F}_w = \Sigma_{i=1}^{\ell} [\hat{\mu}(w)]_{1,i} \cdot \hat{F}_{x_i}$; but
   (b) There exists $y \in Y$ such that
   $\hat{F}_{w \circ \sigma}(y) \neq \Sigma_{i=1}^{\ell} [\hat{\mu}(w)]_{1,i} \cdot \hat{F}_{x_i \circ \sigma}(y)$.
   $x_{\ell+1} \leftarrow w$, $y_{\ell+1} \leftarrow \sigma \circ y$, $X \leftarrow X \cup \{x_{\ell+1}\}$, $Y \leftarrow Y \cup \{y_{\ell+1}\}$, and $\ell \leftarrow \ell + 1$.
   GO TO 2.

The following two claims are used in the proof of correctness. They show that in every iteration of the algorithm, a prefix as required in Step (3) is found, and that as a result the number of independent rows that we have grows by 1.

CLAIM 3.1. *Let z be a counterexample to h found in Step* (3) (*i.e.*, $f(z) \neq h(z)$). *Then, there exists a prefix $w \circ \sigma$ satisfying Conditions* (a) *and* (b).

PROOF. Assume towards a contradiction that no prefix satisfies both (a) and (b). We prove that, for every prefix w of z, Condition (a) is satisfied. That is, $\hat{F}_w = \Sigma_{i=1}^{\ell} [\hat{\mu}(w)]_{1,i} \cdot \hat{F}_{x_i}$. The proof is by induction on the length of w. The induction base is trivial since $\hat{\mu}(\epsilon) = \mathrm{ID}$ (and $x_1 = \epsilon$). For the induction step consider a prefix $w \circ \sigma$. By the induction hypothesis, $\hat{F}_w = \Sigma_{i=1}^{\ell} [\hat{\mu}(w)]_{1,i} \cdot \hat{F}_{x_i}$ which implies (by the assumption that no prefix satisfies both (a) and (b)) that (b) is not satisfied with respect to the prefix $w \circ \sigma$. That is, $\hat{F}_{w \circ \sigma} = \Sigma_{i=1}^{\ell} [\hat{\mu}(w)]_{1,i} \cdot \hat{F}_{x_i \circ \sigma}(y)_0$. By the definition of $\hat{\mu}$ and by the definition of matrix

[10] To check the value of $f(\epsilon)$ we ask a membership query. If $f(\epsilon) = 0$, then we learn f' which is identical to f except that at ϵ it gets some value different than 0. Note that the matrix F' is identical to F in all entries except one and so the rank of F' differs from the rank of F by at most 1. The only change this makes on the algorithm is that before asking EQ we modify the hypothesis h so that its value in ϵ will be 0. Alternatively, we can find a string z such that $f(z) \neq 0$ (by asking EQ(0)) and start the algorithm with $X = \{x_1, x_2\}$ and $Y = \{y_1, y_2\}$ where $x_1 = \epsilon$, $x_2 = z$, $y_1 = \epsilon$ and $y_2 = z$ which gives a 2×2 matrix of full rank.

[11] By the proof of Theorem 2.4, it follows that, if $\ell = r$, then $h \equiv f$. However, we do not need this fact for analyzing the algorithm, and the algorithm does not know r in advance.

multiplication

$$\sum_{i=1}^{\ell}[\hat{\mu}(w)]_{1,i}\cdot\hat{F}_{x_i\circ\sigma}=\sum_{i=1}^{\ell}[\hat{\mu}(w)]_{1,i}\cdot\sum_{j=1}^{\ell}[\hat{\mu}(\sigma)]_{i,j}\cdot\hat{F}_{x_j} \tag{2}$$

$$=\sum_{j=1}^{\ell}\sum_{i=1}^{\ell}[\hat{\mu}(w)]_{1,i}\cdot[\hat{\mu}(\sigma)]_{i,j}\cdot\hat{F}_{x_j}$$

$$=\sum_{j=1}^{\ell}[\hat{\mu}(w\circ\sigma)]_{1,j}\cdot\hat{F}_{x_j}.$$

All together, $\hat{F}_{w\circ\sigma}=\Sigma_{j=1}^{\ell}[\hat{\mu}(w\circ\sigma)]_{1,j}\cdot\hat{F}_{x_j}$, which completes the proof of the induction.

Now, by the induction claim, we get that $\hat{F}_z=\Sigma_{i=1}^{\ell}[\hat{\mu}(z)]_{1,i}\cdot\hat{F}_{x_i}$. In particular, $\hat{F}_z(\epsilon)=\Sigma_{i=1}^{\ell}[\hat{\mu}(z)]_{1,i}\cdot\hat{F}_{x_i}(\epsilon)$ (since $\epsilon\in Y$). However, the left-hand side of this equality is just $f(z)$ while the right-hand side is $h(z)$. Thus, we get $f(z)=h(z)$ which is a contradiction (since z is a counterexample). □

CLAIM 3.2. *Whenever Step* (2) *starts the vectors* $\hat{F}_{x_1},\ldots,\hat{F}_{x_\ell}$ *(defined by the current X and Y) are linearly independent.*

PROOF. The proof is by induction on ℓ. In the first time that Step (2) starts $X=Y=\{\epsilon\}$. By the assumption that $f(\epsilon)\neq 0$, we have a single vector $\hat{F}_\epsilon$ which is not a zero vector, hence the claim holds.

For the induction, assume that the claim holds when Step (2) starts and show that it also holds when Step (3) ends (note that in Step (3) a new vector $\hat{F}_w$ is added and that all vectors have a new coordinate corresponding to $\sigma\circ y$). By the induction hypothesis, when Step (2) starts, $\hat{F}_{x_1},\ldots,\hat{F}_{x_\ell}$ are ℓ linearly independent ℓ-tuples. In particular this implies that when Step (2) starts $\hat{F}_w$ has a unique representation as a linear combination of $\hat{F}_{x_1},\ldots,\hat{F}_{x_\ell}$. Since w satisfies Condition (a), this linear combination is given by $\hat{F}_w=\Sigma_{i=1}^{\ell}[\hat{\mu}(w)]_{1,i}\cdot\hat{F}_{x_i}$. Clearly, when Step (3) ends, $\hat{F}_{x_1},\ldots,\hat{F}_{x_\ell}$ remain linearly independent (with respect to the new Y). However, at this time, $\hat{F}_w$ becomes linearly independent of $\hat{F}_{x_1},\ldots,\hat{F}_{x_\ell}$ (with respect to the new Y). Otherwise, the linear combination must be given by $\hat{F}_w=\Sigma_{i=1}^{\ell}[\hat{\mu}(w)]_{1,i}\cdot\hat{F}_{x_i}$. However, as $w\circ\sigma$ satisfies Condition (b), we get that

$$\hat{F}_w(\sigma\circ y)=\hat{F}_{w\circ\sigma}(y)\neq\sum_{i=1}^{\ell}[\hat{\mu}(w)]_{1,i}\cdot\hat{F}_{x_i\circ\sigma}(y)=\sum_{i=1}^{\ell}[\hat{\mu}(w)]_{1,i}\cdot\hat{F}_{x_i(\sigma\circ y)}$$

which eliminates this linear combination. (Note that $\sigma\circ y$ was added to Y so $\hat{F}$ is defined in all the coordinates which we refer to.) To conclude, when Step (3) ends $\hat{F}_{x_1},\ldots,\hat{F}_{x_\ell},\hat{F}_{x_{\ell+1}}$ (where $x_{\ell+1}=w$) are linearly independent. □

We summarize the analysis of the algorithm by the following theorem. Let m denote the size of the longest counterexample z obtained during the execution of the algorithm. Denote by $\mathrm{M}(r)=O(r^{2.376})$ the complexity of multiplying two

$r \times r$ matrices (see, for example, Knuth [1998, pp. 499–501] for discussion on matrix multiplication).

THEOREM 3.3. *Let $\mathcal{K}$ be a field, and $f: \Sigma^* \rightarrow \mathcal{K}$ be a function such that $r = rank(F)$ (over $\mathcal{K}$). Then, f is learnable by the above algorithm in time $O(|\Sigma| \cdot r \cdot M(r) + m \cdot r^3)$ using r equivalence queries and $O((|\Sigma| + \log m)r^2)$ membership queues.*

PROOF. Claim 3.1 guarantees that the algorithm always proceeds. Since the algorithm halts only if EQ(h) returns YES, the correctness follows.

As for the complexity, Claim 3.2 implies that the number of iterations, and therefore the number of equivalence queries, is at most r (in fact, Theorem 2.4 implies that the number of iterations is exactly r).

The number of MQs asked in Step (2) over the whole algorithm is $(|\Sigma| + 1)r^2$, since for every $x \in X$ and $y \in Y$ we need to ask for the value of $f(x \circ y)$ and the values $f(x \circ \sigma \circ y)$, for all $\sigma \in \Sigma$. To analyze the number of MQs asked in Step (3), we first need to specify the way that the appropriate prefix is found. The naive way is to go over all prefixes of z until finding one satisfying Conditions (a) and (b). A more efficient search can be based upon the following generalization of Claim 3.1: Suppose that for some v, a prefix of z, Condition (a) holds. That is, $\hat{F}_v = \Sigma_{i=1}^{\ell} [\hat{\mu}(v)]_{1,i} \cdot \hat{F}_{x_i}$. Then, there exists $w \circ \sigma$ a prefix of z that extends v and satisfies Conditions (a) and (b) (the proof is identical to the proof of Claim 3.1 except that for the base of induction we use v instead of ϵ). Using the generalized claim, the desired prefix $w \circ \sigma$ can be found using a binary search in $\log |z| \leq \log m$ steps as follows: at the middle prefix v check whether Condition (a) holds. If so, make v the left border for the search. If Condition (a) does not hold for $v = w \circ \sigma$, then, by Eq. (2), Condition (b) holds for v and so v becomes the right border for the search. In each step of the binary search $2\ell \leq 2r$ membership queries are asked (note that the values of $\hat{F}_{x_i}$ and $\hat{F}_{x_i} \circ \sigma$ are known from Step (2)). All together, the number of MQs asked during the execution of the algorithm is $O((\log m + |\Sigma|)r^2)$.

As for the running time, to compute each of the matrices $\hat{\mu}_\sigma$ observe that the matrix whose rows are $\hat{F}_{x_1} \circ \sigma, \ldots, \hat{F}_{x_\ell} \circ \sigma$ is the product of $\hat{\mu}_\sigma$ with the matrix whose rows are $\hat{F}_{x_1}, \ldots, \hat{F}_{x_\ell}$. Therefore, finding $\hat{\mu}_\sigma$ can be done with one matrix inversion, whose cost is also $O(\mathrm{M}(r))$ (see, for example, Cormen et al. [1990, Theorem 31.11]), and one matrix multiplication. Hence, the complexity of Step (2) is $O(|\Sigma| \cdot \mathrm{M}(r))$. In Step (3), the difficult part is to compute the value of $[\hat{\mu}(z)]_1$ for the counterexample z. A simple way to do it is by computing m matrix multiplications for each such z. A better way of doing the computation of Step (3) is by observing that all we need to compute is actually the first row of the matrix $\hat{\mu}(z) = \hat{\mu}_{z_1} \cdot \hat{\mu}_{z_2} \cdots \hat{\mu}_{z_m}$. The first row of this matrix can simply be written as $[\hat{\mu}(z)]_1 = (1, 0, \ldots, 0) \cdot \hat{\mu}(z)$. Thus, to compute this row, we first compute $(1, 0, \ldots, 0) \cdot \hat{\mu}_{z_1}$, then multiply the result by $\hat{\mu}_{z_2}$ and so on. Therefore, this computation can be done by m vector-matrix multiplications, which requires $O(m \cdot r^2)$ time. Notice that this computation also gives us the value $[\mu(w)]_1$ for every prefix w of z. All together, the running time is at most $O(|\Sigma| \cdot r \cdot \mathrm{M}(r) + m \cdot r^3)$. □

The complexity of our algorithm should be compared to the complexity of the algorithm of Bergadano and Varricchio [1996], which uses r equivalence queries,

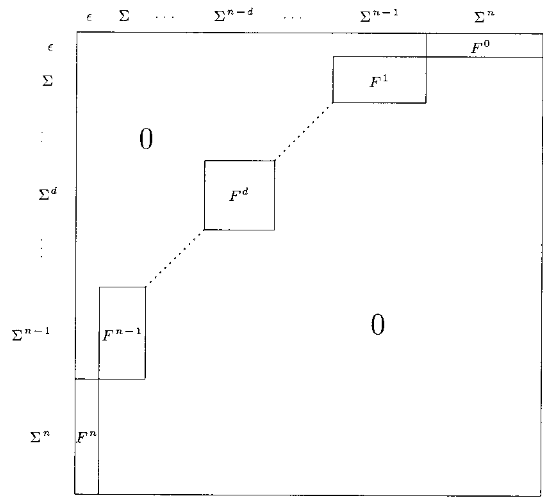

FIG. 1. The Hankel matrix F.

$O(|\Sigma|\ mr^2)$ membership queries, and runs in time $O(|\Sigma|\ m^2\ r^5)$. The algorithm of Ohnishi et al. [1994] uses $r + 1$ equivalence queries, $O((|\Sigma| + m)r^2)$ membership queries, and runs in time $O((|\Sigma| + m)r^4)$. Our algorithm is similar to the algorithm of Ohnishi et al. [1994]; however, we use a binary search in Step (3) and we implement the algorithm more efficiently.

3.1. THE CASE OF FUNCTIONS $f\colon \Sigma^n \rightarrow \mathcal{K}$. In many cases of interest the domain of the target function f is not Σ^* but rather Σ^n for some value n. We view f as a function on Σ^* whose value is 0 for all strings whose length is different from n. We show that in this case the complexity analysis of our algorithm can be further improved. The reason is that in this case the matrix F has a simpler structure. Each row and column is indexed by a string whose length is at most n (alternatively, rows and columns corresponding to longer strings contain only 0 entries). Moreover, for any string x of length $0 \leq d \leq n$ the only nonzero entries in the row F_x correspond to y's of length $n - d$. Denote by F^d the submatrix of F whose rows are strings in Σ^d and its columns are strings in Σ^{n-d} (see Figure 1). Observe that by the structure of F,

$$\operatorname{rank}(F) = \sum_{d=0}^{n} \operatorname{rank}(F^d).$$

Now, to learn such a function f, we use the above algorithm but ask membership queries only on strings of length exactly n (for all other strings we return 0 without actually asking the query) and for the equivalence queries we view the hypothesis h as restricted to Σ^n. The length of counterexamples, in this case, is always n and so $m = n$.

Looking closely at what the algorithm does it follows that since $\hat{F}$ is a submatrix of F, not only are $\hat{F}_{x_1}, \ldots, \hat{F}_{x_\ell}$ always independent vectors (and so $\ell \leq \operatorname{rank}(F)$), but for every d the number of x_i's in X whose length is d is bounded by $\operatorname{rank}(F^d)$. We denote $r^d \triangleq \operatorname{rank}(F^d)$ and $r_{\max} \triangleq \max_{d=0}^{n} r^d$ (clearly,

$r_{\max} \leq r \leq (n+1) \cdot r_{\max}$). The number of equivalence queries remains r as before. The number of membership queries, however, becomes smaller due to the fact that many entries of F are known to be 0. In Step (2), over the whole execution, we ask for every $x \in X$ of length d and every $y \in Y$ of length $n - d$ one membership query on $f(x \circ y)$ and for every $y \in Y$ of length $n - d - 1$ and every $\sigma \in \Sigma$ we ask a membership query on $f(x \circ \sigma \circ y)$. All together, in Step (2), the algorithm asks for every x at most $r_{\max} + |\Sigma| r_{\max}$ membership queries and total of $O(r \cdot r_{\max} |\Sigma|)$ membership queries. In Step (3), in each of the r iterations and each of the $\log n$ search steps, we ask at most $2r_{\max}$ membership queries (again, because most of the entries in each row contain 0's). Thus, the total number of membership queries over the whole algorithm is $O(r \cdot r_{\max}(|\Sigma| + \log n))$.

As for the running time, note that the matrices $\hat{\mu}_\sigma$ also have a very special structure: the only entries (i, j) that are not 0 are those corresponding to vectors $x_i, x_j \in X$ such that $|x_j| = |x_i| + 1$. Hence, inversion and multiplication of such matrices can be done in time $n \cdot \mathrm{M}(r_{\max})$. Therefore, each invocation of Step (2) requires time of $O(|\Sigma| n \cdot \mathrm{M}(r_{\max}))$. Similarly, in $[\hat{\mu}(w)]_1$ the only entries which are not 0 are those corresponding to strings $x_j \in X$ such that $|x_j| = |w|$. Thus, multiplying $[\hat{\mu}(w)]_1$ by a column of $\hat{\mu}_\sigma$ requires $r_{\max}$ time units. Furthermore, we need to multiply at most $r_{\max}$ columns, for the non-zero coordinates in $[\hat{\mu}(w \circ \sigma)]_1$. Therefore, Step (3) takes at most $nr_{\max}^2$ for each counterexample z. All together, the running time is at most $O(nrr_{\max}^2 + |\Sigma| rn \cdot \mathrm{M}(r_{\max})) = O(|\Sigma| rn \cdot \mathrm{M}(r_{\max}))$.

COROLLARY 3.4. *Let $\mathcal{K}$ be a field, and $f\colon \Sigma^n \to \mathcal{K}$ such that $r = rank(F)$ and $r_{max} = max_{d=0}^{n}\, rank(F^d)$ (where rank is taken over $\mathcal{K}$). Then, f is learnable by the above algorithm in time $O(|\Sigma| rn \cdot M(r_{max}))$ using $O(r)$ equivalence queries and $O((|\Sigma| + log\ n) r \cdot r_{\max})$* membership queries.

4. *Positive Results*

In this section, we show the learnability of various classes of functions by our algorithm. This is done by proving that for every function f in the class in question, the corresponding Hankel matrix F has low rank. By Theorem 3.3, this implies the learnability of the class by our algorithm. We next summarize the results of this section. In Section 4.1, we show that the rank of the Hankel matrix of (generalized) polynomials is small. In particular, this implies the learnability of polynomials over fixed finite fields, and bounded degree polynomials over any field. Furthermore, we show that the learnability of generalized polynomials implies the learnability of subclasses of boxes (Section 4.2) and satisfy-$O(1)$ DNF and other subclasses of DNF (Section 4.3). In Section 4.4, we prove that if two functions have Hankel matrices with small rank then the rank of the Hankel matrix of their product is small. We show that this implies the learnability of certain classes of decision trees and a certain subclass of arithmetic circuits of depth two.

In the next paragraph, we mention some immediate results implied by the learnability of multiplicity automata. We first assert that the learnability of multiplicity automata gives a new algorithm for learning deterministic automata and an algorithm for learning unambiguous automata.[12] To see this, define the (i, j) entry of the matrix μ_σ as 1 if the given automaton can move, on letter σ, from state i to state j (otherwise, this entry is 0). In addition, define γ_i to be 1 if i is an accepting state and 0 otherwise (we assume, without loss of generality, that q_1 is the initial state of the given automaton). This defines a multiplicity automaton which computes the characteristic function of the language of the deterministic (or unambiguous) automaton.[13] By Kushilevitz [1997], the class of deterministic automata contains the class of $O(\log n)$-term DNF and in fact the class of all Boolean functions over $O(\log n)$ terms. Hence, all these classes can be learned by our algorithm. We note that if general nondeterministic automata can be learned then this implies the learnability of DNF.

4.1. CLASSES OF POLYNOMIALS. Our first results use the learnability of multiplicity automata to learn various classes of multivariate polynomials. We start with the following claim:

THEOREM 4.1. *Let $p_{i,j}$: $\Sigma \rightarrow \mathcal{K}$ be arbitrary functions of a single variable* $(1 \leq i \leq t,\ 1 \leq j \leq n)$. *Let g_i: $\Sigma^n \rightarrow \mathcal{K}$ be defined by $\Pi_{j=1}^{n} p_{i,j}(z_j)$. Finally, let f: $\Sigma^n \rightarrow \mathcal{K}$ be defined by $f = \Sigma_{i=1}^{t} g_i$. Let F be the Hankel matrix corresponding to f, and F^d the submatrices defined in Section* 3.1. *Then, for every* $0 \leq d \leq n$, *rank*$(F^d) \leq t$.

PROOF. Recall the definition of F^d. Every string $z \in \Sigma^n$ is viewed as partitioned into two substrings $x = x_1 \cdots x_d$ and $y = y_{d+1} \cdots y_n$ (i.e., $z = x \circ y$). Every row of F^d is indexed by $x \in \Sigma^d$; hence, it can be written as a function

$$F_x^d(y) = f(x \circ y) = \sum_{i=1}^{t} \left(\prod_{j=1}^{d} p_{i,j}(x_j) \right) \left(\prod_{j=d+1}^{n} p_{i,j}(y_j) \right).$$

Now, for every x and i, the term $\Pi_{j=1}^{d} p_{i,j}(x_j)$ is just a constant $\alpha_{i,x} \in \mathcal{K}$. This means, that every function $F_x^d(y)$ is a linear combination of the t functions $\Pi_{j=d+1}^{n} p_{i,j}(y_j)$ (one function for each value of i). This implies that rank$(F^d) \leq t$, as needed. □

COROLLARY 4.2. *The class of functions that can be expressed as functions over $GF(p)$ with t summands, where each summand T_i is a product of the form $p_{i,1}(x_1) \cdots p_{i,n}(x_n)$ (and $p_{i,j}$: $GF(p) \rightarrow GF(p)$ are arbitrary functions) is learnable in time poly(n, t, p).*

The above corollary implies as a special case the learnability of polynomials over GF(p). This extends the result of Schapire and Sellie [1996] from multilinear polynomials to arbitrary polynomials. Our algorithm (see Corollary 3.4), for polynomials with n variables and t terms, uses $O(nt)$ equivalence queries and $O(t^2 n \log n)$ membership queries. The special case of the above class–the class of polynomials over GF(2)–was known to be learnable before Schapire and Sellie

[12] A nondeterministic automata is *unambiguous* if, for every $w \in \Sigma^*$, there is at most one accepting path.

[13] We can associate a multiplicity automaton (over the rationals) with every nondeterministic automaton in the same way. However, to learn this automaton we need "multiplicity queries"; that is, a query that on a string w returns the number of accepting paths of the nondeterministic automaton on w.

[1996]. Their algorithm uses $O(nt)$ equivalence queries and $O(t^3n)$ membership queries (which is worse than ours for "most" values of t).

Corollary 4.2 discusses the learnability of a certain class of functions (that includes the class of polynomials) over *finite* fields (the complexity of the algorithm depends on the size of the field). The following theorem extends this result to *infinite* fields, assuming that the functions $p_{i,j}$ are bounded-degree polynomials. It also improves the complexity for learning polynomials over finite fields, when the degree of the polynomials is significantly smaller than the size of the field.

THEOREM 4.3. *The class of functions over a field $\mathcal{K}$ that can be expressed as t summands, where each summand T_i is of the form $p_{i,1}(x_1) \cdots p_{i,n}(x_n)$, and $p_{i,j}$: $\mathcal{K} \rightarrow \mathcal{K}$ are univariate polynomials of degree at most k, is learnable in time poly(n, t, k). Furthermore, if $|\mathcal{K}| \leq nk + 1$ then this class is learnable from membership queries only in time poly(n, t, k) (with small probability of error).*

PROOF. We show that although the field $\mathcal{K}$ may be very large, we can run the algorithm using an alphabet of $k + 1$ elements from the field, $\Sigma = \{\sigma_1, \ldots, \sigma_{k+1}\}$. For this, all we need to show is how the queries are asked and answered. The membership queries are asked by the algorithm, so it will only present queries which are taken from the domain Σ^n.

To simulate an equivalence query, we first have to extend the hypothesis to the domain $\mathcal{K}^n$ and ask an equivalence query with the extended hypothesis. We then get a counterexample in $\mathcal{K}^n$ and we modify it back to Σ^n. To extend the hypothesis to $\mathcal{K}^n$, instead of representing the hypothesis with $|\Sigma|$ matrices $\hat{\mu}(\sigma_1), \ldots, \hat{\mu}(\sigma_{k+1})$ we will represent it with a single matrix $H(x)$ whose entries are degree k univariate polynomials (over $\mathcal{K}$), such that for every $\sigma \in \Sigma$, $H(\sigma) = \hat{\mu}(\sigma)$. To find this H use interpolation in each of its entries. The hypothesis for $z = z_1 \cdots z_n \in \mathcal{K}^n$ is defined as $h(z) \triangleq [H(z_1) \cdots H(z_n)]_1 \cdot \vec{\gamma}$. Now, it is easy to see that both the target function and the hypothesis are degree-k polynomials in each of the n variables. Given a counterexample $w = w_1 \cdots w_n \in \mathcal{K}^n$, we iteratively modify it to be in Σ^n. Assume that we have already modified $w_1, \ldots, w_{i-1}$ to letters from Σ such that

$$h(w) \neq f(w). \tag{3}$$

We modify w_i such that (3) remains true. We first fix $z_j = w_j$ for all $j \neq i$, that is, we consider the univariate polynomials $h(w_1 \cdots w_{i-1} z_i w_{i+1} \cdots w_n)$ and $f(w_1 \cdots w_{i-1} z_i w_{i+1} \cdots w_n)$. Both polynomials are degree-k univariate polynomials of the variable z_i, which disagree when $z_i = w_i$. Since two different univariate polynomials of degree at most k can agree on at most k points, there is a value $\alpha \in \Sigma$ for which the two polynomials disagree. We find such a value α using membership queries, set w_i to α, and proceed to w_{i+1}. We end up with a new counterexample $w \in \Sigma^n$, as desired.

Assume that $\mathcal{K}$ contains at least $nk + 1$ elements, and let $L = \{\sigma_1, \ldots, \sigma_{nk+1}\}$ be a subset of $\mathcal{K}$. We describe a randomized algorithm, which simulates the previous algorithm without using equivalence queries. For a given ϵ, the algorithm fails with probability ϵ, and uses poly$(n, t, \log 1/\epsilon)$ membership queries. We simulate each equivalence query in the previous algorithm using membership queries. To prove the correctness of the simulation, we use the

Schwartz-Zippel Lemma [Schwartz 1980; Zippel 1979], which guarantees that two different polynomials in $z_1, \ldots, z_n$ of degree k (in each variable) can agree on at most $kn|L|^{n-1}$ assignments in L^n. Therefore, if there is a counterexample to our hypothesis and we pick at random, with uniform distribution, an element in L^n then with probability at least $1 - kn/|L| = 1/(kn + 1)$ we get a counterexample. To simulate an equivalence query, we pick at random $O(kn \log tn/\epsilon)$ points in L^n, independently and uniformly, and for each point z we evaluate the hypothesis and compare it to the value $f(z)$ (obtained using a membership query). If we find no counterexample, then we return the answer YES to the equivalence query. If h is not equivalent to f, then with probability at most

$$\left(1 - \frac{1}{kn + 1}\right)^{O(kn \log(tn/\epsilon))} \geq \frac{\epsilon}{tn}$$

none of the points is a counterexample. The algorithm fails if the simulation of one of the tn equivalence queries returned YES although the hypothesis is not equivalent to the target function. This happens with probability at most ϵ. □

An algorithm that learns multivariate polynomials using only membership queries is called an interpolation algorithm.[14] In Ben-Or and Tiwari [1988], it is shown how to interpolate polynomials over infinite fields using only $2t$ membership queries. Algorithms for interpolating polynomials over finite fields are given in Bshouty and Mansour [1995] and Huang and Rao [1996] provided that the fields are "big" enough (in Huang and Rao [1996], the field must contain $\Omega(t^2k + tkn^2)$ elements and, in Bshouty and Mansour [1995], the field must contain $\Omega(kn/\log kn)$ elements). We require that the number of elements in the field is at least $kn + 1$. However, the polynomials we interpolate in Theorem 4.3 have a more general form than in previous papers; in our algorithm each monomial can be a product of arbitrary univariate polynomials of degree k while in the previous papers each monomial is a product of univariate polynomials of degree k with only one term.[15] To complete the discussion we should mention that if the number of elements in the field is less than k then every efficient algorithm must use equivalence queries [Clausen et al. 1991; Roth and Benedek 1991].

4.2. Classes of Boxes. In this section, we consider unions of n-dimensional boxes in $[\ell]^n$ (where $[\ell]$ denotes the set $\{0, 1, \ldots, \ell - 1\}$). A box in $[\ell]^n$ is defined by two corners $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ (in $[\ell]^n$) as follows:

$$B_{a_1, \ldots, a_n, b_1, \ldots, b_n} = \{(x_1, \ldots, x_n) : \forall i,\ a_i \leq x_i \leq b_i\}.$$

We view such a box as a Boolean function that gives 1 for every point in $[\ell]^n$ which is inside the box and 0 to each point outside the box. We start with a claim about a more general class of functions.

[14] See, for example, Ben-Or and Tiwari [1988], Grigoriev et al. [1990], Zippel [1990], Clausen et al. [1991], Roth and Benedek [1991], Bshouty and Mansour [1995], and Huang and Rao [1996]. For more background and references, see Zippel [1993].

[15] For example, the polynomial $(x_1 + 1)(x_2 + 1) \cdots (x_n + 1)$ has a small size in our representation and requires exponential size in the standard sum-of-terms form.

THEOREM 4.4. *Let $p_{i,j}: \Sigma \rightarrow \{0, 1\}$ be arbitrary functions of a single variable $(1 \leq i \leq t, 1 \leq j \leq n)$. Let $g_i: \Sigma^n \rightarrow \{0, 1\}$ be defined by $\Pi_{j=1}^{n} p_{i,j}(z_j)$. Assume that there is no point $x \in \Sigma^n$ such that $g_i(x) = 1$ for more than s functions g_i. Finally, let $f: \Sigma^n \rightarrow \{0, 1\}$ be defined by $f = \bigvee_{i=1}^{t} g_i$. Let F be the Hankel matrix corresponding to f. Then, for every field $\mathcal{K}$ and for every $0 \leq d \leq n$, $rank(F^d) \leq \Sigma_{i=1}^{s} \binom{t}{i}$.*

PROOF. The function f can be expressed as:

$$\begin{aligned} f &= 1 - \prod_{i=1}^{t}(1 - g_i) \\ &= \sum_i g_i - \sum_{i,j}(g_i \wedge g_j) + \cdots + (-1)^{t+1} \sum_{|S|=t} \bigwedge_{i \in S} g_i \\ &= \sum_i g_i - \sum_{i,j}(g_i \wedge g_j) + \cdots + (-1)^{s+1} \sum_{|S|=s} \bigwedge_{i \in S} g_i, \end{aligned}$$

where the last equality is by the assumption that $g_i(x) \neq 0$ for at most s functions g_i (for every point x). Note that the functions g_i are Boolean; therefore, the $\bigwedge$ operation is just the product operation of the field and hence the above equalities hold over every field. Every function of the form $\bigwedge_{i \in S} g_i$ is a product of at most n functions, each one is a function of a single variable. Therefore, applying Theorem 4.1 completes the proof. □

COROLLARY 4.5. *The class of unions of disjoint boxes can be learned in time poly(n, t, ℓ) (where t is the number of boxes in the target function). The class of unions of O(log n) boxes can be learned in time poly(n, ℓ).*

PROOF. Let B be any box and denote the two corners of B by $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$. Define functions (of a single variable) $p_j: [\ell] \rightarrow \{0, 1\}$ to be 1 if $a_j \leq z_j \leq b_j$ $(1 \leq j \leq n)$. Let $g: [\ell]^n \rightarrow \{0, 1\}$ be defined by $\Pi_{j=1}^{n} p_j(z_j)$. That is, $g(z_1, \ldots, z_n)$ is 1 if and only if $(z_1, \ldots, z_n)$ belongs to the box B. Therefore, Corollary 3.4 and Theorem 4.4 imply this corollary. □

Note that the proof does *not* use any particular property of *geometric* boxes and it applies to *combinatorial* boxes as well (a combinatorial box only requires that every x_i is in some arbitrary set $S_i \subseteq [\ell]$). Methods that do use the specific properties of geometric boxes were developed in Beimel and Kushilevitz [1999] and they lead to improved results.

4.3. CLASSES OF DNF FORMULAS. In this section, we present several results for classes of DNF formulas and some related classes. We first consider the following special case of Corollary 4.2 that solves an open problem of Schapire and Sellie [1996].

COROLLARY 4.6. *The class of functions that can be expressed as exclusive-OR of t (not necessarily monotone) monomials is learnable in time poly(n, t).*

While Corollary 4.6 does not refer to a subclass of DNF, it already implies the learnability of disjoint (i.e., satisfy-1) DNF. Also, since DNF is a special case of union of boxes (with $\ell = 2$), we can get the learnability of disjoint DNF from Corollary 4.5. Next we discuss positive results for satisfy-s DNF with larger values of s. The following two important corollaries follow from Theorem 4.4. Note that Theorem 4.4 holds in any field. For convenience (and efficiency), we will use $\mathcal{K} = \mathrm{GF}(2)$.

COROLLARY 4.7. *The class of satisfy-s DNF formulas, for* $s = O(1)$, *is learnable in time poly*(n, t).

COROLLARY 4.8. *The class of satisfy-s, t-term DNF formulas is learnable in time poly*(n) *for the following choices of s and t*: (1) $t = O(\log n)$; (2) $t = \mathrm{polylog}(n)$ *and* $s = O(\log n/\log\log n)$; (3) $t = 2^{O(\log n/\log\log n)}$ *and* $s = O(\log\log n)$.

4.4. CLASSES OF DECISION TREES. As mentioned above, our algorithm efficiently learns the class of disjoint DNF formulas. This in particular includes the class of Decision-trees. By using our algorithm, decision trees of size t on n variables are learnable using $O(tn)$ equivalence queries and $O(t^2 n \log n)$ membership queries. This is better than the best known algorithm for decision trees [Bshouty 1995a] (which uses $O(t^2)$ equivalence queries and $O(t^2 n^2)$ membership queries). In what follows, we consider more general classes of decision trees.

COROLLARY 4.9. *Consider the class of decision trees that compute functions* $f: GF(p)^n \rightarrow GF(p)$ *as follows: each node v contains a query of the form "*$x_i \in S_v$*?", for some* $S_v \subseteq GF(p)$. *If* $x_i \in S_v$, *then the computation proceeds to the left child of v and if* $x_i \notin S_v$ *the computation proceeds to the right child. Each leaf* ℓ *of the tree is marked by a value* $\gamma_\ell \in GF(p)$ *which is the output on all the assignments which reach this leaf. Then, this class is learnable in time poly*$(n, |L|, p)$, *where L is the set of leaves.*

PROOF. Each such tree can be written as $\Sigma_{\ell \in L} \gamma_\ell \cdot g_\ell(x_1, \ldots, x_n)$, where each g_ℓ is a function whose value is 1 if the assignment $(x_1, \ldots, x_n)$ reaches the leaf ℓ and 0 otherwise (note that in a decision tree each assignment reaches a single leaf). Consider a specific leaf ℓ. The assignments that reach ℓ can be expressed by n sets $S_{\ell,1}, \ldots, S_{\ell,n}$ such that the assignment $(x_1, \ldots, x_n)$ reaches the leaf ℓ if and only if $x_j \in S_{\ell,j}$ for all j. Define $p_{\ell,j}(x_j)$ to be 1 if $x_j \in S_{\ell,j}$ and 0 otherwise. Then $g_\ell = \Pi_{j=1}^{n} p_{\ell,j}$. By Corollary 4.2, the result follows. □

The above result implies as a special case the learnability of decision trees with "greater-than" queries in the nodes. This is an open problem of Bshouty [1995a]. Note that every decision tree with "greater-than" queries that computes a boolean function can be expressed as the union of disjoint boxes. Hence, this case can also be derived from Corollary 4.5. The next theorem will be used to learn more classes of decision trees.

THEOREM 4.10. *Let* $g_i: \Sigma^n \rightarrow \mathcal{K}$ *be arbitrary functions* $(1 \le i \le \ell)$. *Let* $f: \Sigma^n \rightarrow \mathcal{K}$ *be defined by* $f = \Pi_{i=1}^{\ell} g_i$. *Let F be the Hankel matrix corresponding to f, and* G_i *be the Hankel matrix corresponding to* g_i. *Then, rank*$(F^d) \le \Pi_{i=1}^{\ell}$ *rank*(G_i^d).

PROOF. For two matrices A and B of the same dimension, the Hadamard product $C = A \odot B$ is defined by $C_{i,j} = A_{i,j} \cdot B_{i,j}$. It is well known that $\mathrm{rank}(C) \le \mathrm{rank}(A) \cdot \mathrm{rank}(B)$. Note that $F^d = \odot_{i=1}^{\ell} G_i^d$; hence, the theorem follows. □

This theorem has some interesting applications. The first application states that arithmetic circuits of depth two with multiplication gate of fan-in $O(\log n)$ at the top level and addition gates with unbounded fan-in in the bottom level are learnable.

COROLLARY 4.11. *Let $\mathcal{C}$ be the class of functions that can be expressed in the following way: Let $p_{i,j}$: $\Sigma \rightarrow \mathcal{K}$ be arbitrary functions of a single variable* $(1 \leq i \leq \ell$, $1 \leq j \leq n)$. *Let $\ell = O(\log n)$ and g_i: $\Sigma^n \rightarrow \mathcal{K}$ $(1 \leq i \leq \ell)$ be defined by $\Sigma_{j=1}^n p_{i,j}(z_j)$. Finally, let f: $\Sigma^n \rightarrow \mathcal{K}$ be defined by $f = \Pi_{i=1}^{\ell} g_i$. Then, $\mathcal{C}$ is learnable in time poly*$(n$, $|\Sigma|)$.

PROOF. Fix some i, and let G be the Hankel matrix corresponding to g_i. Every row of G^d is indexed by $x \in \Sigma^d$; hence, it can be written as a function

$$G_x^d(y) = f(x \circ y) = \sum_{j=1}^{d} p_{i,j}(x_j) + \sum_{j=d+1}^{n} p_{i,j}(y_j).$$

Now, for every x, the sum $\Sigma_{j=1}^d p_{i,j}(x_j)$ is just a constant $\alpha_x \in \mathcal{K}$. This means, that every function $G_x^d(y)$ is a linear combination of the function $\Sigma_{j=d+1}^n p_{i,j}(y_j)$ and the constant function. This implies that rank$(G^d) \leq 2$, and by Theorem 4.10 rank(F^d) = poly(n). □

COROLLARY 4.12. *Consider the class of decision trees of depth s, where the query at each node v is a Boolean function f_v with $r_{max} \leq t$ (as defined in Section* 3.1) *such that* $(t + 1)^s$ = *poly*(n). *Then, this class is learnable in time poly*$(n$, $|\Sigma|)$.

PROOF. For each leaf ℓ, we write a function g_ℓ as a product of s functions as follows: for each node v along the path to ℓ, if we use the edge labeled 1, we take f_v to the product, while, if we use the edge labeled 0, we take $(1 - f_v)$ to the product (note that the value $r_{\max}$ corresponding to $(1 - f_v)$ is at most $t + 1$). By Theorem 4.10, if G_ℓ is the Hankel matrix corresponding to g_ℓ then rank(G_ℓ^d) is at most $(t + 1)^s$. As $f = \Sigma_{\ell \in L} g_\ell$ it follows that rank(F^d) is at most $2^s \cdot (t + 1)^s$ (this is because $|L| \leq 2^s$ and rank$(A + B) \leq$ rank(A) + rank(B)). The corollary follows. □

The above class contains, for example, all the decision trees of depth $O(\log n)$ that contain in each node a term or a XOR of a subset of variables as defined in Kushilevitz and Mansour [1993] (the fact that $r_{\max} \leq 2$ for XOR of a subset of variables follows from the proof of Corollary 4.11).

5. *Negative Results*

The purpose of this section is to study some limitation of the learnability via the automaton representation. We show that our algorithm, as well as any algorithm whose complexity is polynomial in the size of the automaton (such as the algorithms in Bergadano and Varricchio [1996a] and Ohnishi et al. [1994]), does not efficiently learn several important classes of functions. More precisely, we show that these classes contain functions f that have no "small" automaton. By Theorem 2.4, it is enough to prove that the rank of the corresponding Hankel matrix F is "large" over every field $\mathcal{K}$.

Let $0 \leq k \leq n/2$. We define a function $f_{n,k}$: $\{0, 1\}^n \rightarrow \{0, 1\}$ by $f_{n,k}(z) = 1$ iff there exists $1 \leq i \leq k$ such that $z_i = z_{i+k} = 1$. The function $f_{n,k}$ can be expressed as a DNF formula by:

$$z_1 z_{k+1} \vee z_2 z_{k+2} \vee \cdots \vee z_k z_{2k} .$$

Note that this formula is *read-once, monotone* and has k terms.

First, observe that the rank of the Hankel matrix corresponding to $f_{n,k}$ equals the rank of F, the Hankel matrix corresponding to $f_{2k,k}$. It is also clear that $\text{rank}(F) \geq \text{rank}(F^k)$ (recall that F^k is the submatrix of F whose rows and columns are indexed by strings of length exactly k). We now prove that $\text{rank}(F^k) \geq 2^k - 1$. To do so, we consider the complement matrix D_k (obtained from F^k by switching 0's and 1's), and prove by induction on k that $\text{rank}(D_k) = 2^k$. Consider the (x, y) entry of D_k where $x = x_1 \cdots x_{k-1} x_k$ and $y = y_1 \cdots y_{k-1} y_k$. Its value is 0 if and only if there is an i such that $x_i = y_i = 1$. Hence, if $x_k = y_k = 1$, then the entry is zero and otherwise it equals to the (x', y') of D_{k-1}, where $x' = x_1 \cdots x_{k-1}$ and $y' = y_1 \cdots y_{k-1}$. Thus,

$$D_1 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \qquad D_k = \begin{pmatrix} D_{k-1} & D_{k-1} \\ D_{k-1} & 0 \end{pmatrix}.$$

This implies that $\text{rank}(D_1) = 2$ and $\text{rank}(D_k) = 2 \cdot \text{rank}(D_{k-1})$ which implies $\text{rank}(D_k) = 2^k$. It follows that $\text{rank}(F^k) \geq 2^k - 1$, since $D_k = J - F^k$ (where J is the all-1 matrix).[16] Using the functions $f_{n,k}$ we can now prove the main theorem of this section:

THEOREM 5.1. *The following classes are not learnable in time polynomial in n and the formula size using multiplicity automata* (*over any field* $\mathcal{K}$):

(1) *DNF.*
(2) *Monotone DNF.*
(3) *2-DNF.*
(4) *Read-once DNF.*
(5) *k-term DNF, for* $k = \omega(\log n)$.
(6) *Satisfy-s DNF, for* $s = \omega(1)$.
(7) *Read-j satisfy-s DNF, for* $j = \omega(1)$ *and* $s = \Omega(\log n)$.

Some of these classes are known to be learnable by other methods (monotone DNF [Angluin 1988], read-once DNF [Angluin et al. 1993; Aizenstein and Pitt 1991; Pillaipakkamnatt and Raghavan 1995] and 2-DNF [Valiant 1984]), some are natural generalizations of classes known to be learnable as automata ($O(\log n)$-term DNF [Blum and Rudich 1995; Bshouty 1995a; 1997; Kushilevitz 1997], and satisfy-s DNF for $s = O(1)$ (Corollary 4.7)) or by other methods (read-j satisfy-s for $js = O(\log n/\log\log n)$ [Blum et al. 1994]) and the learnability of some of the others is still an open problem.

[16] In fact, the function $f'_{n,k} = z_1 z_n \vee z_2 z_{n-1} \vee \cdots \vee z_k z_{n-k+1}$ has similar properties to $f_{n,k}$ and can be shown to have rank $\Omega(2^k \cdot n)$, hence slightly improving the results below.

PROOF. Observe that $f_{n,n/2}$ belongs to each of the classes DNF, monotone DNF, 2-DNF, read-once DNF and that by the above argument every automaton for it has size at least $2^{n/2}$. This shows (1)–(4).

For every $k = \omega(\log n)$, the function $f_{n,k}$ has exactly k-terms and every automaton for it has size at least $2^k = 2^{\omega(\log n)}$ which is super-polynomial. This proves (5).

For $s = \omega(1)$, consider the function $f_{n,s\log n}$. Every automaton for it has size at least $2^{s \log n} = n^{\omega(1)}$, which is super-polynomial. We now show that the function $f_{n,s\log n}$ has a small satisfy-s DNF representation. For this, partition the indices $1, \ldots, k = s \log n$ into s sets of $\log n$ indices. For each set S there is a formula on $2 \log n$ variables which is 1 iff there exists $i \in S$ such that $z_i = z_{i+k} = 1$. Moreover, there is such a formula which is satisfy-1 (i.e., disjoint) DNF, and it has n^2 terms (this is the standard DNF representation). The disjunction of these s formulas gives a satisfy-s DNF with sn^2 terms. This proves (6).

Finally, for $j = \omega(1)$ and $s = \Omega(\log n)$ let $k = s \log j = \omega(\log n)$. As before, the function $f_{n,k}$ requires an automaton of super-polynomial size. On the other hand, by partitioning the variables into s disjoint sets of $\log j$ variables as above (and observing that in the standard DNF representation each variable appears $2^{\log j} = j$ times) this function is a read-j satisfy-s DNF. This proves (7). □

In what follows, we wish to strengthen the previous negative results. The motivation is that in the context of automata there is a fixed order on the characters of the string. However, in general (and in particular for functions over Σ^n) there is no such "natural" order. Indeed, there are important functions such as disjoint DNF that are learnable as automata using any order of the variables. On the other hand, there are functions for which certain orders are much better than others. For example, the function $f_{n,k}$ requires an automaton of size exponential in k when the standard order is considered, but if instead we read the variables in the order $1, k + 1, 2, k + 2, 3, k + 3, \ldots$, then there is a small (even deterministic) automaton for it (of size $O(n)$). As an additional example, *every* read-once formula has a "good" order (the order of leaves in a tree representing the formula).

Our goal is to show that even if we had an oracle that could give us a "good" (not necessarily the best) order of the variables (or if we could somehow learn such an order) then still some of the above classes cannot be learned as automata. This is shown by exhibiting a function that has no "small" automaton in any order of the variables. To show this, we define a function $g_{n,k}$: $\{0, 1\}^n \to \{0, 1\}$ (where $3k \leq n$) as follows: Denote the input variables for $g_{n,k}$ as $w_0, \ldots, w_{k-1}, z_0, \ldots, z_{n'-1}$ where $n' = n - k$. The function $g_{n,k}$ outputs 1 iff there exists t such that $w_t = 1$ and

$$(*) \qquad \exists 0 \leq i \leq k - 1 \qquad \text{such that } z_{(i+t)\bmod k} = z_{i+k} = 1.$$

Intuitively, $g_{n,k}$ is similar to $f_{n,k}$ but instead of comparing the first k variables to the next k variables we first apply a "cyclic shift" to the first k variables by t.[17]

[17] The rank method used to prove that every automaton for $f_{n,k}$ is "large" is similar to the rank method of *communication complexity*. The technique we use next is also similar to methods used in *variable partition communication complexity*. For background see, for example, Lengauer [1990] and Kushilevitz and Nisan [1997].

First, we show how to express $g_{n,k}$ as a DNF formula. For a fixed t, define a function $g_{n',k,t}$ on $z_0, \ldots, z_{n'-1}$ to be 1 iff (*) holds. Observe that $g_{n',k,t}$ is isomorphic to $f_{n',k}$ and so it is representable by a DNF formula (with k terms of size 2). Now, we write $g_{n,k} = \bigvee_{t=0}^{k-1}(w_t \wedge g_{n',k,t})$. Therefore, $g_{n,k}$ can be written as a monotone, read-k, DNF of k^2 terms each of size 3.

We now show that, for *every* order π on the variables, the rank of the matrix corresponding to $g_{n,k}$ is large. For this, it is sufficient to prove that for some value t the rank of the matrix corresponding to $g_{n',k,t}$ is large, since this is a submatrix of the matrix corresponding to $g_{n,k}$ (to see this fix $w_t = 1$ and $w_j = 0$ for all $j \neq t$). As before, it is sufficient to prove that for some t the rank of $g_{2k,k,t}$ is large. The main technical issue is to choose the value of t. For this, look at the order that π induces on $z_0, \ldots, z_{2k-1}$ (ignoring $w_0, \ldots, w_{k-1}$). Look at the first k indices in this order and assume, without loss of generality, that at least half of them are from $\{0, \ldots, k - 1\}$ (hence, out of the last k indices at least half are from $\{k, \ldots, 2k - 1\}$). Denote by A the set of indices from $\{0, \ldots, k - 1\}$ that appear among the first k indices under the order π. Denote by B the set of indices i such that $i + k$ appears among the last k indices under the order π. Both A and B are subsets of $\{0, \ldots, k - 1\}$ and by the assumption, $|A|$, $|B| \geq k/2$. Define $A_t = \{i | (i + t \bmod k) \in A\}$. We now show that for some t the size of $A_t \cap B$ is $\Omega(k)$. For this, write

$$\sum_{t=0}^{k-1} |A_t \cap B| = \sum_{j \in B} |\{t | j \in A_t\}| = |A| \cdot |B| \geq \frac{k^2}{4}.$$

Let t_0 be such that $S = A_{t_0} \cap B$ has size $|S| \geq k/4$. Denote by G the matrix corresponding to g_{2k,k,t_0}. In particular, let G' be the submatrix of G with rows that are all strings x of length k (according to the order π) whose bits not in S are fixed to 0's and with columns that are all strings y of length k whose bits which are not of the form $i + k$, for some $i \in S$, are fixed to 0's. This matrix is the same matrix obtained in the proof for $f_{2k,|S|}$ whose rank is therefore at least $2^{k/4} - 1$.

COROLLARY 5.2. *The following classes are not learnable in time polynomial in n and the formula size using automata (over any field $\mathcal{K}$) even if the best order is known*:

(1) *DNF.*

(2) *Monotone DNF.*

(3) *3-DNF.*

(4) *k-term DNF, for* $k = \omega(\log^2 n)$.

(5) *Satisfy-s DNF, for* $s = \omega(1)$.

PROOF. Observe that $g_{n,n/3}$ belongs to each of the classes DNF, monotone DNF, and 3-DNF and that by the above argument, for every order on the variable, every automaton for it has size $2^{n/12}$. This shows (1)–(3).

For every $k = \omega(\log^2 n)$, the function $g_{n,\sqrt{k}}$ has at most k-terms and, for every order on the variable, every automaton for it has size $2^{\sqrt{k}/2} = 2^{\omega(\log n)}$ which is super-polynomial. This proves (4).

For (5), consider the function

$$h_{n,k} = \bigvee_{t=0}^{k-1} (\bar{w}_1 \wedge \bar{w}_{t-1} \wedge w_t \wedge g_{n',k,t}).$$

By the same arguments as above, for every order on the variables, the rank of the matrix corresponding to $h_{n,k}$ is large (at least $2^{k/4}$). For $s = w(1)$, consider the function $h_{n,s\log n'}$. For every order on the variables, every automaton for it has size $2^{s\log n'} = n^{w(1)}$, which is super-polynomial. We now show that the function $h_{n,s \log n'}$ has a small satisfy-s DNF representation. By the same arguments as in the proof of Theorem 5.1, every function $g_{n',\, s \log n,t}$ has a small satisfy-s DNF formula. Since every assignment can satisfy at most one function $g_{n',\, s \log n',t}$, the function $h_{n,s \log n'}$ has a small satisfy-s DNF formula as well. This proves (5). □

REFERENCES

AIZENSTEIN, H., AND PITT, L. 1991. Exact learning of read-twice DNF formulas. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 170–179.

AIZENSTEIN, H., AND PITT, L. 1992. Exact learning of read-*k* disjoint DNF and not-so-disjoint DNF. In *Proceedings of 5th Annual ACM Workshop on Computational Learning Theory* (Pittsburgh, Pa., July 27–29). ACM, New York, pp. 71–76.

ANGLUIN, D. 1987a. Learning *k*-term DNF formulas using queries and counterexamples. Tech. Rep. YALEU/DCS/RR-559. Dept. Computer Science; Yale University.

ANGLUIN, D. 1987b. Learning regular sets from queries and counterexamples. *Inf. Comput. 75*, 87–106.

ANGLUIN, D. 1988. Queries and concept learning. *Mach. Learn. 2*, 4, 319–342.

ANGLUIN, D., HELLERSTEIN, L., AND KARPINSKI, M. 1993. Learning read-once formulas with queries. *J. ACM 40*, 185–210.

AUER, P. 1993. On-line learning of rectangles in noisy environments. In *Proceedings of 6th Annual ACM Conference on Computational Learning Theory*, (Santa Cruz, Calif., July 26–28). ACM, New York, pp. 253–261.

BEIMEL, A., BERGADANO, F., BSHOUTY, N. H., KUSHILEVITZ, E., AND VARRICCHIO, S. 1996. On the applications of multiplicity automata in learning. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 349–358.

BEIMEL, A., AND KUSHILEVITZ, E. 1998. Learning boxes in high dimension. *Algorithmica, 22*, 76–90.

BEN-OR, M., AND TIWARI, P. 1988. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (Chicago, Ill., May 2–4). ACM, New York, pp. 301–309.

BERGADANO, F., CATALANO, D., AND VARRICCHIO, S. 1996. Learning sat-*k*-DNF formulas from membership queries. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing* (Philadelphia, Pa., May 22–24). ACM, New York, pp. 126–130.

BERGADANO, F., AND VARRICCHIO, S. 1996a. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM J. Comput. 25*, 6, 1268–1280.

BERGADANO, F., AND VARRICCHIO, S. 1996b. Learning behaviors of automata from shortest counterexamples. In *EuroCOLT '95, Lecture Notes in Artificial Intelligence*, vol. 904. Springer-Verlag, New York, pp. 380–391.

BERSTEL, J., AND REUTENAUER, C. 1988. *Rational Series and Their Languages*. In *EATCS monograph on Theoretical Computer Science*, vol. 12. Springer-Verlag, New York.

BLUM, A., KHARDON, R., KUSHILEVITZ, E., PITT, L., AND ROTH, D. 1994. On learning read-*k*-satisfy-*j* DNF. In *Proceedings of 7th Annual ACM Conference on Computational Learning Theory* (New Brunswick, N.J., July 12–15). ACM, New York, pp. 110–117.

BLUM, A., AND RUDICH, S. 1995. Fast learning of *k*-term DNF formulas with queries. *J. Comput. Syst. Sci. 51*, 3, 367–373.

BREIMAN, L., FRIEDMAN, J. H., OLSHEN, R. A., AND STONE, C. J. 1984. *Classification and Regression Trees*. Wadsworth International Group.

BSHOUTY, N. H. 1995a. Exact learning via the monotone theory. *Inf. Comput. 123*, 1, 146–153.

BSHOUTY, N. H. 1995b. A note on learning multivariate polynomials under the uniform distribution. In *Proceedings of 8th Annual. ACM Conference on Computational Learning Theory* (Santa Cruz, Calif., July 5–8). ACM, New York, pp. 79–82.

BSHOUTY, N. H. 1997. Simple learning algorithms using divide and conquer. *Comput. Complex. 6*, 174–194.

BSHOUTY, N. H., AND MANSOUR, Y. 1995. Simple learning algorithms for decision trees and multivariate polynomials. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, New York, pp. 304–311.

BSHOUTY, N. H., TAMON, C., AND WILSON, D. K. 1998. Learning matrix functions over rings. *Algorithmica 22,* 91–111.

CARLYLE, J. W., AND PAZ, A. 1971. Realization by stochastic finite automaton. *J. Comput. Syst. Sci. 5*, 26–40.

CHEN, Z., AND MAASS, W. 1992. On-line learning of rectangles. In *Proceedings of 5th Annual ACM Workshop on Computational Learning Theory*, (Pittsburgh, Pa., July 27–29). ACM, New York, pp. 16–28.

CLAUSEN, M., DRESS, A., GRABMEIER, J., AND KARPINSKI, M. 1991. On zero-testing and interpolation of k-sparse multivariate polynomials over finite fields. *Theoret. Comput. Sci. 84*, 2, 151–164.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L. 1990. *Introduction to algorithms*. MIT Press, Cambridge, Mass., and McGraw-Hill, New York.

EILENBERG, S. 1974. *Automata, Languages and Machines*, vol. A. Academic Press, Orlando, Fla.

FLIESS, M. 1974. Matrices de Hankel. *J. Math. Pures Appl., 53*, 197–222. (Erratum in vol. 54, 1975.)

GOLDBERG, P. W., GOLDMAN, S. A., AND MATHIAS, H. D. 1994. Learning unions of boxes with membership and equivalence queries. In *Proceedings of 7th Annual ACM Conference on Computational Learning Theory* (New Brunswick, N.J., July 12–15) ACM, New York, pp. 148–207.

GRIGORIEV, D. Y., KARPINSKI, M., AND SINGER, M. F. 1990. Fast parallel algorithms for sparse multivariate polynomial interpolation over finite fields. *SIAM J. Comput. 19*, 6, 1059–1063.

HANCOCK, T. R. 1991. Learning 2μ DNF formulas and $k\mu$ decision trees. In *Proceedings of 4th Annual ACM Workshop on Computational Learning Theory* (Santa Cruz, Calif., Aug. 5–7). ACM, New York, pp. 199–209.

HUANG, M. A., AND RAO, A. J. 1996. Interpolation of sparse multivariate polynomials over large finite fields with applications. In *Proceedings of the 7th Annual ACM–SIAM Symposium on Discrete Algorithms*. ACM, New York, pp. 508–517.

JACKSON, J. C. 1997. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *J. Comput. Syst. Sci. 55*, 3, 414–440.

KEARNS, M. J., AND VALIANT, L. G. 1994. Cryptographic limitations on learning Boolean formulae and finite automata. *J. ACM 41*, 1 (Jan.), 67–95.

KEARNS, M. J., AND VAZIRANI, U. V. 1994. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, Mass.

KHARITONOY, M. 1993. Cryptographic hardness of distribution-specific learning. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing* (San Diego, Calif., May 16–18). ACM, New York, pp. 372–381.

KNUTH, D. E. 1998. *The Art of Computer Programming. Vol. 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Mass.

KUSHILEVITZ, E. 1997. A simple algorithm for learning $O(\log n)$-term DNF. In *Inform. Process. Lett. 61*, 6, 289–292.

KUSHILEVITZ, E., AND MANSOUR, Y. 1993. Learning decision trees using the Fourier spectrum. *SIAM J. Comput. 22*, 6, 1331–1348.

KUSHILEVITZ, E., AND NISAN, N. 1997. *Communication Complexity*. Cambridge University Press, Cambridge, Mass.

LANG, K. J. 1992. Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of 5th Annual ACM Workshop on Computational Learning Theory* (Pittsburgh, Pa., July 27–29). ACM, New York, pp. 45–52.

LENGAUER, T. 1990. VLSI theory. In *Handbook of Theoretical Computer Science*, vol. A, chap. 16, J. van Leeuwen, ed. Elsevier, Amsterdam, The Netherlands and The MIT Press, Cambridge, Mass., pp. 835–868.

MAASS, W., AND TURAN, G. 1989. On the complexity of learning from counterexamples. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Los Alamitos, Calif., pp. 262–273.

MAASS, W., AND TURÁN, G. 1994. Algorithms and lower bounds for on-line learning of geometrical concepts. *Mach. Learning 14*, 251–269.

MAASS, W., AND WARMUTH, M. K. 1998. Efficient learning with virtual threshold gates. *Inf. Comput. 141*, 1, 66–83.

OHNISHI, H., SEKI, H., AND KASAMI, T. 1994. A polynomial time learning algorithm for recognizable series. *IEICE Trans. Inf. Syst. E77-D*, (10) (5), 1077–1085.

PILLAIPAKKAMNATT, K., AND RAGHAVAN, V. 1995. Read-twice DNF formulas are properly learnable. *Inf. Comput. 122*, 2, 236–267.

QUINLAN, J. R. 1986. Induction of decision trees. *Mach. Learn. 1*, 81–106.

QUINLAN, J. R. 1993. *C4.5: Programs for machine learning*. Morgan-Kaufmann.

RIVEST, R. L., AND SCHAPIRE, R. E. 1993. Inference of finite automata using homing sequences. *Inf. Comput. 103*, 299–347.

ROTH, R. M., AND BENEDEK, G. M. 1991. Interpolation and approximation of sparse multivariate polynomials over $GF(2)$. *SIAM J. Comput*. 20, 2, 291–314.

SCHAPIRE, R. E., AND SELLIE, L. M. 1996. Learning sparse multivariate polynomials over a field with queries and counterexamples. *J. Comput. Syst. Sci. 52*, 2, 201–213.

SCHWARTZ, J. T. 1980. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM 27*, 701–717.

SHÜTZENBERGER, M. P. 1961. On the definition of a family of automata. *Inf. Control 4*, 245–270.

TRAKHTENBROT, B. A., AND BARZDIN, Y. M. 1973. *Finite Automata: Behavior and Synthesis*. North-Holland, Amsterdam, The Netherlands.

VALIANT, L. G. 1984. A theory of the learnable. *Commun. ACM 27*, 11 (Nov.), 1134–1142.

VALIANT, L. G. 1985. Learning disjunctions of conjunctions. In *Proceedings of the International Joint Conference of Artificial Intelligence*. Morgan-Kaufmann, Los Angeles, Calif., pp. 560–566.

ZIPPEL, R. E. 1979. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Manipulation* (*EUROSAM '79*). Lecture Notes in Computer Science, vol. 72. Springer-Verlag, New York, pp. 216–226.

ZIPPEL, R. E. 1990. Interpolating polynomials from their values. *J. Symbolic Comput. 9*, 375–403.

ZIPPEL, R. E. 1993. Efficient Polynomial Computation. Kluwer Academic Publishers, Hingham, Mass.

RECEIVED JANUARY 1997; REVISED AUGUST 1999; ACCEPTED SEPTEMBER 1999