On Optimal Learning Algorithms for Multiplicity Automata

Laurence Bisht, Nader H. Bshouty, and Hanna Mazzawi

Department of Computer Science
Technion, Haifa 32000, Israel
{bisht, bshouty, hanna}@cs.technion.ac.il

Abstract. We study polynomial time learning algorithms for Multiplicity Automata (MA) and Multiplicity Automata Function (MAF) that minimize the access to one or more of the following resources: Equivalence queries, Membership queries or Arithmetic operations in the field $\mathcal{F}$. This is in particular interesting when access to one or more of the above resources is significantly more expensive than the others.

We apply new algebraic approach based on Matrix Theory to simplify the algorithms and the proofs of their correctness. We improve the arithmetic complexity of the problem and argue that it is almost optimal. Then we prove tight bound for the minimal number of equivalence queries and almost (up to *log* factor) tight bound for the number of membership queries.

1 Introduction

In computational learning theory, one of the interesting problems studied in the literature is learning the classes of Multiplicity Automata (MA) and Multiplicity Automata Function (MAF) over any field from membership (substitution) and equivalence queries [20, 12, 6, 13, 7, 4, 5, 11, 10]. This class includes many interesting classes such as: decision trees, disjoint DNF, $O(\log n)$-term DNF, multivariate polynomials, DFA, boxes and more. In all the algorithms in the literature, it is assumed that the cost of all the resources are the same. In practice, one resource may be more expensive than the others. For example, if the field is the reals then arithmetic operation is not one unit step.

In this paper we study polynomial time learning algorithms for Multiplicity Automata that minimize the access to one or more of the following resources: Equivalence queries, Membership queries or Arithmetic operations in the field $\mathcal{F}$. First, we improve the arithmetic complexity of the problem and argue that it is optimal. Then prove tight bound for the minimal number of equivalence queries and almost (up to *log* factor) tight bound for the number of membership queries.

We summarize the contributions of the paper in the following:

1. **Matrix Approach:** Representing MA in algebraic structure enables using the theory of matrices. We show that the representation of any MA is unique up to similarity of matrices. This gives simple algorithms and analysis.
2. **Arithmetic Complexity:** With this new algebraic approach we use techniques from algebraic complexity to improve the arithmetic complexity of

G. Lugosi and H.U. Simon (Eds.): COLT 2006, LNAI 4005, pp. 184–198, 2006.

the problem. We also introduce a new hypothesis class called the Extended Multiplicity Automata class (EMA) and learn MA from this class with arithmetic complexity that is *independent* of the alphabet size. We show that the arithmetic complexity in this algorithm is optimal in the sense that it is equal to the arithmetic complexity of computing the target function in all the counterexamples received by the algorithm.

3. **Equivalence Query Complexity**: We prove a lower bound for the number of equivalence queries. Then we give a polynomial time learning algorithm for MA that is optimal in the equivalence query complexity.
4. **Membership Query Complexity**: We give a lower bound for the number of queries which is almost tight (up to *log* factor). This gives an almost tight lower bound for the number of membership queries.
5. **Results for MAF:** In the full paper, we also obtain results similar to the above for MAF. We introduce a new representation of a MAF called the compressed MAF and use it to show learnability that obtains optimal number of equivalence queries and almost optimal number of membership queries and arithmetic operations for MAF.

2 Preliminaries

2.1 Concept Classes

Let $\Sigma = \{\sigma_1, \ldots, \sigma_t\}$ be a finite alphabet of size t and $\mathcal{F}$ be a field. A *Multiplicity Automaton Function* (MAF) with an alphabet Σ over a field $\mathcal{F}$ with n variables is a function $f : \Sigma^n \to \mathcal{F}$ of the form $f(w_1 w_2 \cdots w_n) = \Lambda_1^{(w_1)} \cdots \Lambda_n^{(w_n)}$ where for every $\sigma \in \Sigma$ and every i, $\Lambda_i^{(\sigma)}$ is $s_i \times s_{i+1}$ matrix with entries from $\mathcal{F}$ and $s_1 = s_{n+1} = 1$. We define the *size of* f *at level* i as $\text{size}_i(f) = s_i$, the *width* of f is $\text{size}_{\max}(f) = \max_i s_i$ and the *size* of f, $\text{size}(f)$, is $\sum_i s_i$. See full paper for a graph representation of MAF.

A *Multiplicity Automaton* (MA) with an alphabet Σ over a field $\mathcal{F}$ is a function $f : \Sigma^* \to \mathcal{F}$ of the form $f(w_1 w_2 \cdots w_m) = \beta \Lambda^{(w_1)} \Lambda^{(w_2)} \cdots \Lambda^{(w_m)} \gamma^T$ where for each $\sigma \in \Sigma$, $\Lambda^{(\sigma)}$ is $s \times s$ matrix and β, γ are s vectors over $\mathcal{F}$. We define $\Lambda^{(\varepsilon)} = I$ the identity matrix and $\Lambda^{(\sigma w)} = \Lambda^{(\sigma)} \Lambda^{(w)}$ for every $\sigma \in \Sigma$ and string w. Then we can write $f(w) = \beta \Lambda^{(w)} \gamma^T$ for any string w. We call s the *size* of the MA f. See full paper for a graph representation of MA.

An *Extended Multiplicity Automaton* (EMA) with an alphabet Σ over a field $\mathcal{F}$ is a function $f : \Sigma^* \to \mathcal{F}$ of the form $f(w_1 w_2 \cdots w_m) = \beta \Lambda^{(w_1)} \hat{\Lambda} \Lambda^{(w_2)} \hat{\Lambda} \cdots \Lambda^{(w_m)} \hat{\Lambda} \gamma^T$, where $\Lambda^{(\sigma)}$, $\hat{\Lambda}$ are $s \times s$ matrices and β, γ are s vectors. Obviously, an EMA is an MA with $\tilde{\Lambda}^{(\sigma)} = \Lambda^{(\sigma)} \hat{\Lambda}$. See full paper for a graph representation of EMA.

2.2 Properties of MA

In this section we give some properties of MA

The next Theorem shows that if we have $2s$ strings $x_1, \ldots, x_s, y_1, \ldots, y_s$ such that the matrix $[f(x_i \cdot y_j)]_{i,j}$ (the i,jth entry is $f(x_i \cdot y_j)$) is non-singular, then we can construct the MA for f.

Theorem 1. *Let $f(x) = \beta\Lambda^{(x)}\gamma^T$ be an MA of size s. Let $x_1, \ldots, x_s, y_1, \ldots, y_s$ be strings in Σ^* such that $[f(x_i \cdot y_j)]_{i,j}$ is non-singular. Then $f(x) = \beta_0\Lambda_0^{(x)}\gamma_0^T$ where*

$$\Lambda_0^{(\sigma)} = M^{(\sigma)}N^{-1}, \quad \gamma_0 = (f(x_1), \ldots, f(x_s)), \quad \beta_0 = (f(y_1), \ldots, f(y_s))N^{-1}$$

and $M^{(\sigma)} = [f(x_i \cdot \sigma \cdot y_j)]_{i,j}$ and $N = [f(x_i \cdot y_j)]_{i,j}$.

Proof. Let

$$K = \begin{pmatrix} \beta\Lambda^{(x_1)} \\ \beta\Lambda^{(x_2)} \\ \vdots \\ \beta\Lambda^{(x_s)} \end{pmatrix} \text{ and } L = (\Lambda^{(y_1)}\gamma^T | \cdots | \Lambda^{(y_s)}\gamma^T).$$

Now we have $N = KL$, $M^{(\sigma)} = K\Lambda^{(\sigma)}L$, $(f(x_1), \ldots, f(x_s))^T = K\gamma^T$ and $(f(y_1), \ldots, f(y_s)) = \beta L$. Since $N = KL$ is non-singular, K and L are non-singular. Now it straightforward to show that for any word w we have $\beta_0\Lambda_0^{(w)}\gamma_0^T = \beta\Lambda^{(w)}\gamma^T = f(w)$. ■

Now, we show that the MA representation is unique up to similarity

Lemma 1. *Let $f(x) = \beta_1\Lambda_1^{(x)}\gamma_1^T$ and $g(x) = \beta_2\Lambda_2^{(x)}\gamma_2^T$ be two MAs of size s. We have $f(x) \equiv g(x)$ if and only if there is a non-singular matrix J such that $\beta_2 = \beta_1 J^{-1}$, $\gamma_2^T = J\gamma_1^T$ and $\Lambda_2^{(\sigma)} = J\Lambda_1^{(\sigma)}J^{-1}$ for every $\sigma \in \Sigma$.*

Proof. The "If" part of the Lemma is straightforward. For the "only if" part, define K_1, L_1 and K_2, L_2, as defined in the proof of Theorem 1, for $\beta_1\Lambda_1^{(x)}\gamma_1^T$ and $\beta_2\Lambda_2^{(x)}\gamma_2^T$, respectively. Then $N = K_1L_1 = K_2L_2$ and $M^{(\sigma)} = K_1\Lambda_1^{(\sigma)}L_1 = K_2\Lambda_2^{(\sigma)}L_2$. Now for $J = K_2^{-1}K_1$ we have $J = L_2L_1^{-1}$ and $\Lambda_2^{(\sigma)} = J\Lambda_1^{(\sigma)}J^{-1}$. ■

A similar result is proved for MAF in the full paper.

2.3 The Learning Model

Our learning model is the *exact* learning model [2, 17]. In this model a *teacher* has a *target function* f that the *learner* (*learning algorithm*) wants to learn from queries. In the *equivalence query*, the learner gives the teacher a hypothesis h. The teacher returns either *yes*, signifying that h is equivalent to f, or *no* with a *counterexample*, which is an assignment $(b, f(b))$ such that $h(b) \neq f(b)$. In the *membership query*, the learner gives the teacher an assignment a. The teacher returns $f(a)$.

We say that the learner *learns* a class of functions C, if for every function $f \in C$, the learner outputs a hypothesis h that is equivalent to f. The goal of the learner is to learn in polynomial time where "polynomial time" means polynomial in the *size* of the shortest representation of f and the longest counterexample returned by the teacher.

3 The Algorithm and Its Analysis

In this section we introduce a simple algorithm similar to the one in [4] with the theory of matrices. Then we give a simple proof for its correctness. We will assume $|\Sigma| > 1$. See the full paper for unary alphabet.

3.1 The Learning Algorithm for MA

The algorithm initially asks EQ(0) and receives a counterexample x_1. Then it defines $X = \{x_1\}$ and $Y = \{y_1 = \varepsilon\}$. At stage ℓ it uses two sets of strings $X = \{x_1, x_2, \ldots, x_\ell\}$ and $Y = \{y_1 = \varepsilon, y_2, \ldots, y_\ell\}$ where $N(X,Y) = N = [f(x_i \cdot y_j)]_{i,j}$ is a non-singular matrix. Then the algorithm defines the hypothesis defined in Theorem 1. That is, $h(x) = \beta_0 \Lambda_0^{(x)} \gamma_0^T$ where

$$\Lambda_0^{(\sigma)} = M^{(\sigma)} N^{-1},\ \gamma_0 = (f(x_1), \ldots, f(x_\ell)),\ \beta_0 = (f(y_1), \ldots, f(y_\ell)) N^{-1}$$

and

$$M^{(\sigma)} = [f(x_i \cdot \sigma \cdot y_j)]_{i,j}$$

for every $\sigma \in \Sigma$. Notice that $\gamma_0^T = N e_1^T$ where e_i is the ith unit vector.

Now the algorithm asks equivalence query with $h(x)$ and receives a counterexample $z \in \Sigma^*$ where $f(z) \neq h(z)$. The algorithm then finds a prefix $w \cdot \sigma$ of z, where $\sigma \in \Sigma$, such that (see Fact 1 in the next subsection)

$$(f(w \cdot y_1), \ldots, f(w \cdot y_\ell)) = \beta_0 \Lambda_0^{(w)} N \tag{1}$$

and

$$(f(w \cdot \sigma \cdot y_1), \ldots, f(w \cdot \sigma \cdot y_\ell)) \neq \beta_0 \Lambda_0^{(w \cdot \sigma)} N, \tag{2}$$

and adds $x_{\ell+1} = w$ to X and $y_{\ell+1} = \sigma \cdot y_{i_0}$ to Y where i_0 is any entry that satisfies $f(w \cdot \sigma \cdot y_{i_0}) \neq \beta_0 \Lambda_0^{(w \cdot \sigma)} N e_{i_0}^T$. Such entry exists because of (2). Then it goes to stage $\ell + 1$.

3.2 Correctness of the Algorithm

We first show that such $w \cdot \sigma$, that satisfies (2), exists and then show that the new matrix $N(\hat{X}, \hat{Y})$, where $\hat{X} = \{x_1, \ldots, x_\ell, x_{\ell+1}\}$ and $\hat{Y} = \{y_1, \ldots, y_\ell, y_{\ell+1}\}$, is non-singular.

Fact 1. *If $f(z) \neq h(z)$ then there is a prefix $w \cdot \sigma$ for z that satisfies (1).*

Proof. It is enough to show that the first equality in (1) is true for $w = \varepsilon$ and the second inequality in (2) is true for $w = z$. For the prefix $w = \varepsilon$ we have $(f(y_1), \ldots, f(y_\ell)) = \beta_0 N = \beta_0 \Lambda_0^{(\varepsilon)} N$. Now for $w = z$, $f(z \cdot y_1) = f(z) \neq h(z) = \beta_0 \Lambda_0^{(z)} \gamma_0^T = \beta_0 \Lambda_0^{(z)} N e_1^T$. ∎

Now we show

Fact 2. *The new matrix $\hat{N} = N(\hat{X}, \hat{Y})$ is non-singular.*

Proof. We have $\hat{X} = X \cup \{w\}$ and $\hat{Y} = Y \cup \{\sigma \cdot y_{i_0}\}$. Therefore

$$\hat{N} = \begin{pmatrix} N & M^{(\sigma)} e_{i_0}^T \\ \beta_0 \Lambda_0^{(w)} N & f(w \cdot \sigma \cdot y_{i_0}) \end{pmatrix}.$$

Now since $M^{(\sigma)} = \Lambda_0^{(\sigma)} N$,

$$\begin{pmatrix} I & 0 \\ \beta_0 \Lambda_0^{(w)} & -1 \end{pmatrix} \hat{N} = \begin{pmatrix} N & M^{(\sigma)} e_{i_0}^T \\ 0 & \beta_0 \Lambda_0^{(w\cdot\sigma)} N e_{i_0}^T - f(w \cdot \sigma \cdot y_{i_0}) \end{pmatrix},$$

and $\beta_0 \Lambda_0^{(w\cdot\sigma)} N e_{i_0}^T - f(w \cdot \sigma \cdot y_{i_0}) \neq 0$, the rank of $\hat{N}$ is $\ell + 1$. ■

3.3 The Complexity of the Algorithm

In all the results, m is the longest counterexample received by the learner and the time complexity is linear in the number of queries and arithmetic operations.

A straightforward algebraic computation gives the same query and arithmetic complexity as in [4]. In [4], Beimel et. al. proved the following.

Theorem 2. *Let $\mathcal{F}$ be a field, and $f : \Sigma^* \to \mathcal{F}$ be an MA of size s. Then f is learnable as MA from $s+1$ equivalence queries and $O((|\Sigma| + \log m)s^2)$ membership queries in*

$$O(|\Sigma| s M(s) + m s^3) = O(|\Sigma| s^{3.37} + m s^3)$$

arithmetic operations. Here, $M(s)$ is the complexity of $s \times s$ matrix multiplication.

In the next section we improve the arithmetic complexity to $O(|\Sigma| s^3 + m s^3)$ using MA hypothesis and then to $O(m s^3)$ using EMA hypothesis.

4 Almost Optimal Arithmetic Complexity

In this section we prove the following

Theorem 3. *Let $\mathcal{F}$ be a field, and $f : \Sigma^* \to \mathcal{F}$ be an MA of size s. Then f is learnable as EMA from $s+1$ equivalence queries and $O((|\Sigma| + \log m)s^2)$ membership queries with*

$$O(m s^3)$$

arithmetic operations.

Notice that the arithmetic complexity in the Theorem is independent of how large is the alphabet. The arithmetic complexity is almost optimal in the following sense: It is known, [9], that the equivalence query complexity of any polynomial time learning algorithm for MA is at least $\tilde{\Omega}(s)$. Computing the target

hypothesis in $\tilde{\Omega}(s)$ strings of length m takes $\tilde{\Omega}(ms^3)$ arithmetic operations in the field. So the optimality is in the sense that the arithmetic complexity of the algorithm is within logarithmic factor of the arithmetic complexity for computing the target hypothesis in all the counterexamples received in the algorithm.

Proof of Theorem 3. At stage ℓ the algorithm asks equivalence query with a hypothesis $h(x)$ and receives a counterexample $z \in \Sigma^*$. Then for every prefix w of z it computes $\beta_0 \Lambda_0^{(w)}$ and $\beta_0 \Lambda_0^{(w)} N$. This can be done in $O(m\ell^2)$ arithmetic operations. Then the algorithm does a binary search to find a prefix that satisfies (1). This takes $\ell \log m$ membership queries. Then it builds $\hat{N}$. Notice that all the entries of $\hat{N}$ are already known from previous computations. To build $\hat{M}^{(\sigma)}$ for all $\sigma \in \Sigma$ it needs to ask membership queries to find $f(x_{\ell+1} \cdot \sigma \cdot y_i)$, $f(x_i \cdot \sigma \cdot y_{\ell+1})$ and $f(x_{\ell+1} \cdot \sigma \cdot y_{\ell+1})$. This takes $(2\ell+1)|\Sigma|$ membership queries. By Lemma 2 below $\hat{N}^{-1}$ and each $\hat{\Lambda}_0^{(\sigma)}$ can be computed with $O(\ell^2)$ arithmetic operations. Therefore it needs $O(|\Sigma|\ell^2)$ arithmetic operations to compute all $\hat{\Lambda}_0^{(\sigma)}$. Finally, to compute $\hat{\beta}_0$ it needs $O(\ell^2)$ arithmetic operations.

This gives arithmetic complexity $O(|\Sigma|s^3 + ms^3)$ and the algorithm outputs MA. In the case where the output can be EMA, we can omit the step that computes $\Lambda_0^{(\sigma)}$ for every σ and output the EMA

$$f(w) = \hat{\beta}_0 M^{(w_1)} N^{-1} M^{(w_2)} N^{-1} \cdots M^{(w_{|w|})} N^{-1} \hat{\gamma}_0^T.$$

This gives the result. ∎

The results for MAF are in the full paper.

Lemma 2. *Let N and M be two $\ell \times \ell$ matrices with entries from $\mathcal{F}$ and $\Lambda = MN^{-1}$. Let $u, \lambda \in \mathcal{F}^\ell$ be such that $u = \lambda N$. Let*

$$\hat{N} = \begin{pmatrix} N & v^T \\ u & \xi \end{pmatrix}, \hat{M} = \begin{pmatrix} M & p^T \\ q & \eta \end{pmatrix}$$

where $\hat{N}$ is nonsingular matrix where $v, q, p \in \mathcal{F}^\ell$ and $\xi, \eta \in \mathcal{F}$. Then

$$\hat{N}^{-1} = \frac{1}{\omega} \begin{pmatrix} \omega N^{-1} - (N^{-1}v^T)\lambda & N^{-1}v^T \\ \lambda & -1 \end{pmatrix}$$

where $\omega = \lambda v^T - \xi$ and

$$\hat{M}\hat{N}^{-1} = \frac{1}{\omega} \begin{pmatrix} \omega\Lambda - (\Lambda v^T)\lambda + p^T\lambda & \Lambda v^T - p^T \\ \omega(qN^{-1}) - ((qN^{-1})v^T)\lambda + \eta\lambda & (qN^{-1})v^T - \eta \end{pmatrix}.$$

From the definitions of MA and EMA we have

Fact 3. *1. For an MA f of size s and a string $a \in \Sigma^*$, $f(a)$ can be computed in $|a|s^2$ arithmetic operations.*

2. For an EMA f of size s and a string $a \in \Sigma^*$, $f(a)$ can be computed in $2|a|s^2$ arithmetic operations.

Can we compute $f(a)$ faster? Proving lower bounds for the number of arithmetic complexity of problems is one of the hardest tasks in algebraic complexity. Techniques used today give only lower bounds that are linear in the number of distinct variables of the problem. For example, the best lower bound for matrix multiplication is $2.5s^2$ for any field, [3] and $3s^2$ for the binary field, [21]. In the problem of computing $f(a)$ for any MA f and any $a \in \Sigma^*$, the number of distinct variables is $\min(|\Sigma|, |a|)s^2 + 2s$, which is the number of the entries of $\Lambda^{(\sigma)}$, β and γ. To the best of our knowledge, no better lower bound is known for this problem. This bound does not match the upper bound in Fact 3, which we believe is optimal up to some log factor.

In the full paper, the following slightly better upper bound is proved

Fact 4. *We have*

For an MA or EMA f of size s and t strings $a_1, \ldots, a_t$, $a_i \in \Sigma^*$ and $|a_i| \le m$ for every i, $f(a_i)$ can be computed in

$$O\left(\frac{ts^2 m \log|\Sigma|}{\log(ts^2 m/M(s))}\right)$$

arithmetic operations when $(ts^2m)/M(s) > 2$ and $O(ts^2m)$ arithmetic operations otherwise.

5 An Optimal Arithmetic Complexity

In this section we define a compressed MA that, with the results of the previous section, will further improve the arithmetic complexity of the algorithm. The bound we achieve here matches the arithmetic complexity of computing the target in all the s counterexamples received in the algorithm.

We first add a new symbol $\flat$ to the alphabet Σ and call it *blank* . For this symbol $\Lambda_0^{(\flat)} = I$ the identity matrix. This means that for any $w \in (\Sigma \cup \{\flat\})^*$, $f(w)$ is equal to $f(\hat{w})$ where $\hat{w}$ is w without the blanks.

Let $\hat{f}$ be a function $\hat{f} : \Sigma^* \to \mathcal{F}$. Define a function $f : (\Sigma \cup \{\flat\})^* \to \mathcal{F}$ where $f(w) = \hat{f}(\hat{w})$. It is clear that the size of $\hat{f}$ is equal to the MA size of f.

For an alphabet Σ we define the alphabet $\Sigma^{[\ell]} = \{[w_1 w_2 \cdots w_\ell] \mid w_i \in \Sigma\}$. Define an operator $\phi : \Sigma^{[\ell]} \to \Sigma^\ell$ where $\phi([w_1 w_2 \cdots w_\ell]) = w_1 w_2 \cdots w_\ell$. For a function $f : \Sigma^* \to \mathcal{F}$ we define the *ℓ-compressed function* $f^{[\ell]} : (\Sigma^{[\ell]})^* \to \mathcal{F}$ as follows: $f^{[\ell]}(u_1 u_2 \cdots u_t) = f(\phi(u_1)\phi(u_2)\cdots\phi(u_t))$.

It is easy to see that the MA size of $f^{[\ell]}$ is at most the MA size of f. Just define $\Lambda_0^{([w_1 w_2 \cdots w_\ell])} = \Lambda_0^{(w_1)} \Lambda_0^{(w_2)} \cdots \Lambda_0^{(w_\ell)}$. It is also easy to see that membership queries and equivalence queries to $f^{[\ell]}$ can be simulated using membership queries and equivalence queries to f and if we learn $f^{[\ell]}$ we can construct f in linear time.

Using the above representation with $\ell = (\epsilon(\log m + \log s))/(\log|\Sigma|)$ we have

Lemma 3. *Let $\mathcal{F}$ be a field, and $f : \Sigma^* \to \mathcal{F}$ be an MA of size s. Then for any constant ϵ, f is learnable as EMA from $s+1$ equivalence queries and $m^\epsilon s^{2+\epsilon}$ membership queries with*

$$O\left(\frac{ms^3 \log |\Sigma|}{\log m + \log s}\right)$$

arithmetic operations.

In the full paper we show that this bound is true even if the learner does not know m and s.

6 Almost Optimal Query Complexity

In this section we prove a lower bound for the number of queries and show that the main algorithm is optimal up to $(\log m)/|\Sigma|$.

We first prove the following

Theorem 4. *Any learning algorithm that learns MA of size at most s* over any field *must ask at least $|\Sigma|s^2 - s^2$ queries.*

Proof. Let $\mathcal{F}$ be any field and $\Sigma = \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$. Define the field extension $\mathcal{K} = \mathcal{F}(\{z_{i,j,k}\})$ of $\mathcal{F}$ with $s^2|\Sigma|$ algebraically independent elements $\{z_{i,j,k}|\ i,j = 1,\ldots,s\ ,\ k = 1,\ldots,|\Sigma|\}$. Denote by z the vector $((z_{i,j,k})_{i,j,k})$ (that contains $z_{i,j,k}$ in some order). Consider any algorithm A that learns MA over any field. Then A, in particular, learns MA over $\mathcal{K}$.

Consider $\Lambda^{(\sigma_k)} = [z_{i,j,k}]_{i,j}$, $\gamma = \beta = e_1$ and the MA $f(x) = \beta\Lambda^{(x)}\gamma^T$. We run the algorithm A on the target f. Let $f(w_1), \ldots, f(w_t)$ be the queries asked to the membership query or received by the equivalence query in A. Notice that for all $r = 1, \ldots, t$, $f(w_r) = \beta\Lambda^{(w_r)}\gamma^T = p_r(z)$ for some multivariate polynomial p_r. The algorithm finds $\beta_0, \Lambda_0^{(\sigma)}$ and γ_0 where $f(x) = \beta_0\Lambda_0^{(x)}\gamma_0^T$. By Lemma 1 there is a non-singular matrix K with entries from $\mathcal{K}$ such that $\Lambda^{(\sigma_k)} = K^{-1}\Lambda_0^{(\sigma_k)}K$ for every k. Since the entries of $\Lambda^{(\sigma_k)}$, $k = 1, \ldots, |\Sigma|$ are algebraically independent and are generated from the entries of K and $p_r(z)$ we must have: The number of entries of K, plus, the number of the polynomials p_r is at least the number of entries of $\Lambda^{(\sigma_k)}$, $k = 1, \ldots, |\Sigma|$. This gives $t + s^2 \geq |\Sigma|s^2$ which implies the result. ∎

Notice that the lower bound in Theorem 4 is true for learning algorithms that are independent of the ground field $\mathcal{F}$, i.e., learning algorithms that learn MA for any field. We now show that any algorithm that learns MA in some specific field $\mathcal{F}$ requires the same number of queries (up to constant factor).

Theorem 5. *Let $\mathcal{F}$ be any field. Any algorithm that learns MA of size at most s over $\mathcal{F}$ must ask at least $(|\Sigma|s^2 - s^2 - O(s))/4 = \Omega(|\Sigma|s^2)$ queries.*

Proof. Assume w.l.o.g that s is even and $r = s/2$. Let $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{t-1}\}$. Let A be an algorithm that learns MA of size at most s over $\mathcal{F}$. Notice here that A may not learn MA over larger fields, so the technique used in the previous Theorem cannot be applied here. We will show an adversarial strategy that forces A to ask at least $|\Sigma|r^2 - r^2$ queries. Consider the $r \times r$ matrix

$$\Lambda_0 = \begin{pmatrix} 0\,1\,0\,0\cdots 0 \\ 0\,0\,1\,0\cdots 0 \\ 0\,0\,0\,1\cdots 0 \\ \vdots\,\vdots\,\vdots\,\vdots\,\ddots\,\vdots \\ 0\,0\,0\,0\cdots 1 \\ 1\,0\,0\,0\cdots 0 \end{pmatrix}.$$

Define the subset $C \subset$MA where for each $f \in C$ we have

$$\Lambda^{(\sigma_0)} = \begin{pmatrix} \Lambda_0 & 0 \\ 0 & \Lambda_0 \end{pmatrix} \text{ and } \Lambda^{(\sigma_i)} = \begin{pmatrix} \Lambda_i & \Lambda_i \\ -\Lambda_i & -\Lambda_i \end{pmatrix}$$

for $i > 0$. where each Λ_i is $r \times r$ 0-1 matrix, and $\beta = e_r$ and $\gamma = e_1$ are $(2r)$-vectors.

We now show the following properties of the functions in C

Claim. We have: for every $f \in C$

1. $f(x \cdot \sigma_0^i \cdot y) = f(x \cdot \sigma_0^{i \bmod r} \cdot y)$ for any two strings x and y.
2. $f(\sigma_0^i \cdot \sigma_k \cdot \sigma_0^{r-j+1}) = \Lambda_k[i,j]$ for $1 \le i,j \le r$.
3. $f(x) = 0$ for every x that contains more than one symbol from $\{\sigma_1, \ldots, \sigma_{t-1}\}$.

Proof of Claim. (1) follows from the fact that $\Lambda_0^r = I$, the identity matrix.

To prove (2), let $1 \le i,j \le r$. Notice that $\beta(\Lambda^{(\sigma_0)})^i = e_i$ and $(\Lambda^{(\sigma_0)})^j\gamma^T = e_{r-j+1}^T$. Also, for any $s \times s$ matrix Z we have $\beta(\Lambda^{(\sigma_0)})^i Z(\Lambda^{(\sigma_0)})^{r-j+1}\gamma^T = Z_{i,j}$. Then

$$f(\sigma_0^i \cdot \sigma_k \cdot \sigma_0^{r-j+1}) = \beta\Lambda^{(\sigma_0^i \cdot \sigma_k \cdot \sigma_0^{r-j+1})}\gamma^T = \beta(\Lambda^{(\sigma_0)})^i\Lambda^{(\sigma_k)}(\Lambda^{(\sigma_0)})^{r-j+1}\gamma^T = \Lambda_k[i,j].$$

Now $(\Lambda^{(\sigma_0)})^i\Lambda^{(\sigma_k)}(\Lambda^{(\sigma_0)})^j$ is of the form $\begin{pmatrix} \Delta & \Delta \\ -\Delta & -\Delta \end{pmatrix}$ and multiplying two matrices of such form gives the zero matrix. This implies (3). ■

Now when the algorithm asks membership query with a string x. If x contains two symbols from $\{\sigma_1, \ldots, \sigma_{t-1}\}$ then the adversary returns 0 and if $x = \sigma_0^i$ then the algorithm returns 1 if $i \bmod r = 1$ and 0 otherwise. In those cases the learner does not gain any information about the function. When the algorithm asks membership query with $x = \sigma_0^i \cdot \sigma_k \cdot \sigma_0^j$ then the adversary answers with arbitrary value from $\{0,1\}$. The learner then knows one of the entries of $\Lambda^{(\sigma_k)}$. If the learner asks again a membership query with $x = \sigma_0^i \cdot \sigma_k \cdot \sigma_0^j$ the adversary returns the same answer.

If the algorithm asks equivalence query with any hypothesis h, the adversary finds some entry $\Lambda_k[i,j]$ that the learner doesn't know from previous query and returns the string $\sigma_0^i \cdot \sigma_k \cdot \sigma_0^{r-j+1}$ as a counterexample.

Notice that each query determines exactly one entry in $\Lambda_k[i,j]$. Since we have $|\Sigma|r^2 - r^2$ entries the algorithm will ask at least $|\Sigma|r^2 - r^2$ queries. ■

The lower bound for the MAF is in the full paper.

7 Optimal Equivalence Query Complexity

In this section we prove a tight bound for the number of equivalence queries of any polynomial time algorithm for MA.

It is known from [16] that under certain cryptographic assumptions DFA (and therefore MA) is not learnable from equivalence queries only in polynomial time. Similar to the technique used in [8, 9] one can prove the following:

Theorem 6. *Any polynomial time learning algorithm for MA must ask at least*

$$\Omega\left(\frac{s\log|\Sigma|}{\log s}\right)$$

equivalence queries.

This proves that the main algorithm in this paper in almost optimal. We now show that this lower bound is tight.

7.1 The Algorithm

Let $f(x) = \beta\Lambda^{(x)}\gamma^T$ be the target MA. The algorithm begins by asking MQ(ε), we assume, without loss of generality, that $f(\varepsilon) \neq 0$, [4]. The algorithm then defines $X = \{x_1 = \varepsilon\}$ and $Y = \{y_1 = \varepsilon\}$. As in the main algorithm, the algorithm maintains two sets of strings $X = \{x_1, x_2, \ldots, x_\ell\}$ and $Y = \{y_1, y_2, \ldots, y_\ell\}$. Also, $N(X,Y) = N = [f(x_i{\cdot}y_j)]_{i,j}$ is non-singular matrix. For some fixed integer $k \geq 3$ the algorithm defines $h(x) = \beta_0\Lambda_0^{(x)}\gamma_0^T$, where $\Lambda_0^{(\sigma)} = M^{(\sigma)}N^{-1}$,

$$\gamma_0 = (f(x_1), f(x_2), \ldots, f(x_\ell)),\ \beta_0 = (f(y_1), f(y_2), \ldots, f(y_\ell))N^{-1}$$

and $M^{(\tau)} = [f(x_i \cdot \tau \cdot y_j)]_{i,j}$ for $\tau \in \Sigma^{\leq k} \stackrel{\text{def}}{=} \cup_{i\leq k}\Sigma^i$. Note that $\gamma_0 = Ne_1^T$ and $\beta_0 = e_1$.

Now instead of asking an equivalence query, the algorithm performs an internal checking step. It tries to find a counterexample using membership queries. The algorithm checks for all i,j and $\tau \in \Sigma^{\leq k}$ whether $f(x_i{\cdot}\tau{\cdot}y_j) = h(x_i{\cdot}\tau{\cdot}y_j)$ by asking membership queries. If for some i,j and $\tau \in \Sigma^{\leq k}$, $f(x_i{\cdot}\tau{\cdot}y_j) \neq h(x_i{\cdot}\tau{\cdot}y_j)$ then the algorithm has found a counterexample and it proceeds as in the main algorithm. Otherwise, the algorithm asks an equivalence query and receives a counterexample $z = z_1 z_2 \cdots z_{|z|}$.

Our goal is to show that the algorithm uses z to generate k additional independent rows and columns in N. For $k = (\log s)/\log|\Sigma|$ we obtain a polynomial time learning algorithm that asks at most

$$\frac{s}{k} = O\left(\frac{s\log|\Sigma|}{\log s}\right)$$

equivalence queries.

Fact 5. *Before the algorithm asks an equivalence query $M^{(\tau)} = \Lambda_0^{(\tau)}N$ for every $\tau \in \Sigma^{\leq k}$.*

Proof. Let

$$K = \begin{pmatrix} \beta_0 \Lambda_0^{(x_1)} \\ \beta_0 \Lambda_0^{(x_2)} \\ \vdots \\ \beta_0 \Lambda_0^{(x_\ell)} \end{pmatrix} \text{ and } L = (\Lambda_0^{(y_1)} \gamma_0^T | \cdots | \Lambda_0^{(y_\ell)} \gamma_0^T).$$

For every $\sigma \in \Sigma$ we have

$$M^{(\sigma)} = K \Lambda_0^{(\sigma)} L \text{ and } N = KL.$$

Thus,

$$\Lambda_0^{(\sigma)} = M^{(\sigma)} N^{-1} = K \Lambda_0^{(\sigma)} K^{-1}.$$

Now, for every $\tau = \tau_1 \tau_2 \cdots \tau_{|\tau|} \in \Sigma^{\leq k}$, we have

$$\begin{aligned} M^{(\tau)} &= [f(x_i \cdot \tau \cdot y_j)]_{i,j} = [h(x_i \cdot \tau \cdot y_j)]_{i,j} = K \Lambda_0^{(\tau)} L = K \Lambda_0^{(\tau_1)} \Lambda_0^{(\tau_2)} \cdots \Lambda_0^{(\tau_{|\tau|})} L \\ &= K \Lambda_0^{(\tau_1)} K^{-1} \cdot K \Lambda_0^{(\tau_2)} K^{-1} \cdot \cdots \cdot K \Lambda_0^{(\tau_{|\tau|})} K^{-1} \cdot KL = \Lambda_0^{(\tau)} N. \end{aligned}$$ ■

Next, the algorithm searches for the minimal length prefix w_1 of z, such that for some $\sigma_1 \in \Sigma$

$$(f(w_1 \cdot \sigma_1 \cdot y_1), f(w_1 \cdot \sigma_1 \cdot y_2), \ldots, f(w_1 \cdot \sigma_1 \cdot y_\ell)) \neq \beta_0 \Lambda_0^{(w_1 \cdot \sigma_1)} N.$$

Such prefix exists since $f(z \cdot y_1) = f(z) \neq h(z) = \beta_0 \Lambda_0^{(z)} \gamma_0^T = \beta_0 \Lambda_0^{(z)} N e_1^T$. Since w_1 is minimal, we get that $(f(w_1 \cdot y_1), f(w_1 \cdot y_2), \ldots, f(w_1 \cdot y_\ell)) = \beta_0 \Lambda_0^{(w_1)} N$. By Fact 5 for all $\tau \in \Sigma^{\leq 2}$, $M^{(\tau)} = \Lambda_0^{(\tau)} N$ and thus,

$$(f(\tau \cdot y_1), f(\tau \cdot y_2), \ldots, f(\tau \cdot y_\ell)) = e_1 M^{(\tau)} = e_1 \Lambda_0^{(\tau)} N = \beta_0 \Lambda_0^{(\tau)} N, \qquad (3)$$

and therefore we have $|w_1| > 1$.

Now the algorithm searches for minimal length prefix w_2 of w_1, such that for some $\sigma_2 \in \Sigma$

$$(f(w_2 \cdot \sigma_2 \sigma_1 \cdot y_1), f(w_2 \cdot \sigma_2 \sigma_1 \cdot y_2), \ldots, f(w_2 \cdot \sigma_2 \sigma_1 \cdot y_\ell)) \neq \beta_0 \Lambda_0^{(w_2 \cdot \sigma_2 \sigma_1)} N.$$

Again by Fact 5 for all $\tau \in \Sigma^{\leq 3}$ it follows that $M^{(\tau)} = \Lambda_0^{(\tau)} N$ and, as in (3) we conclude that $|w_2| > 1$.

The algorithm repeats the above construction k times. Denote by $\sigma^{(i)} = \sigma_i \sigma_{i-1} \cdots \sigma_1$. In the jth iteration it searches for a minimal length prefix w_j of w_{j-1} such that for some $\sigma_j \in \Sigma$

$$(f(w_j \cdot \sigma_j \cdot \sigma^{(j-1)} \cdot y_1), \ldots, f(w_j \cdot \sigma_j \cdot \sigma^{(j-1)} \cdot y_\ell)) \neq \beta_0 \Lambda_0^{(w_j \cdot \sigma_j \cdot \sigma^{(j-1)})} N.$$

After k iterations, the algorithm has a set of strings $W = \{w_1, w_2, \ldots, w_k\}$ and strings $\sigma^{(i)}$ for $1 \leq i \leq k$.

Lemma 4. *For every $1 \leq i \leq k$ and $i > j$ it follows that*

$$(f(w_i \cdot \sigma^{(j)} \cdot y_1), f(w_i \cdot \sigma^{(j)} \cdot y_2), \ldots, f(w_i \cdot \sigma^{(j)} \cdot y_\ell)) = \beta_0 \Lambda_0^{(w_i \cdot \sigma^{(j)})} N$$

Proof. Suppose on the contrary that there exists i and j, such that, $i > j$ and

$$(f(w_i \cdot \sigma^{(j)} \cdot y_1), f(w_i \cdot \sigma^{(j)} \cdot y_2), \ldots, f(w_i \cdot \sigma^{(j)} \cdot y_\ell)) \neq \beta_0 \Lambda_0^{(w_i \cdot \sigma^{(j)})} N,$$

since $i > j$ then w_i is a prefix of w_j, contradiction to the minimality of w_j. ■

To conclude, we found a set of strings $W = \{w_1, w_2, \ldots, w_k\}$ and strings $\sigma^{(i)}$ that satisfy the following properties for every $1 \leq i \leq k$ and $i > j$:

$$(f(w_i \cdot \sigma^{(j)} \cdot y_1), f(w_i \cdot \sigma^{(j)} \cdot y_2), \ldots, f(w_i \cdot \sigma^{(j)} \cdot y_\ell)) = \beta_0 \Lambda_0^{(w_i \cdot \sigma^{(j)})} N \qquad (4)$$

and

$$(f(w_i \cdot \sigma^{(i)} \cdot y_1), f(w_i \cdot \sigma^{(i)} \cdot y_2), \ldots, f(w_i \cdot \sigma^{(i)} \cdot y_\ell)) \neq \beta_0 \Lambda_0^{(w_i \cdot \sigma^{(i)})} N. \qquad (5)$$

Now the algorithm adds W to X, that is, $\hat{X} = X \bigcup W$, and $\sigma^{(j)} \cdot y_{i_j}$ to Y where i_j is any entry that satisfies

$$f(w_j \cdot \sigma^{(j)} \cdot y_{i_j}) \neq \beta_0 \Lambda_0^{(w_j \cdot \sigma^{(j)})} N e_{i_j}^T, \qquad (6)$$

that is, $\hat{Y} = Y \bigcup \{\sigma^{(1)} \cdot y_{i_1}, \sigma^{(2)} \cdot y_{i_2}, \ldots, \sigma^{(k)} \cdot y_{i_k}\}$.

We now prove that the new matrix $\hat{N} = \hat{N}(\hat{X}, \hat{Y})$

$$\hat{N} = \begin{pmatrix} N & M^{(\sigma^{(1)})} e_{i_1}^T & M^{(\sigma^{(2)})} e_{i_2}^T & \ldots & M^{(\sigma^{(k)})} e_{i_k}^T \\ \beta_0 \Lambda_0^{(w_1)} N & f(w_1 \cdot \sigma^{(1)} \cdot y_{i_1}) & f(w_1 \cdot \sigma^{(2)} \cdot y_{i_2}) & \ldots & f(w_1 \cdot \sigma^{(k)} \cdot y_{i_k}) \\ \beta_0 \Lambda_0^{(w_2)} N & f(w_2 \cdot \sigma^{(1)} \cdot y_{i_1}) & f(w_2 \cdot \sigma^{(2)} \cdot y_{i_2}) & \ldots & f(w_2 \cdot \sigma^{(k)} \cdot y_{i_k}) \\ \beta_0 \Lambda_0^{(w_3)} N & f(w_3 \cdot \sigma^{(1)} \cdot y_{i_1}) & f(w_3 \cdot \sigma^{(2)} \cdot y_{i_2}) & \ldots & f(w_3 \cdot \sigma^{(k)} \cdot y_{i_k}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta_0 \Lambda_0^{(w_k)} N & f(w_k \cdot \sigma^{(1)} \cdot y_{i_1}) & f(w_k \cdot \sigma^{(2)} \cdot y_{i_2}) & \ldots & f(w_k \cdot \sigma^{(k)} \cdot y_{i_k}) \end{pmatrix}$$

is non-singular.

By Fact 5 above, $M^{(\tau)} = \Lambda_0^{(\tau)} N$ for all $\tau \in \Sigma^{\leq k}$ and by (5) and (6) for all j:

$$\beta_0 \Lambda^{(w_j \cdot \sigma^{(j)})} N e_{i_j}^T - f(w_j \cdot \sigma^{(j)} \cdot y_{i_j}) \neq 0,$$

we get that:

$$\begin{pmatrix} I & 0 & 0 & 0 & \ldots & 0 \\ \beta_0 \Lambda_0^{(w_1)} & -1 & 0 & 0 & \ldots & 0 \\ \beta_0 \Lambda_0^{(w_2)} & 0 & -1 & 0 & \ldots & 0 \\ \beta_0 \Lambda_0^{(w_3)} & 0 & 0 & -1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta_0 \Lambda_0^{(w_k)} & 0 & 0 & 0 & \ldots & -1 \end{pmatrix} \hat{N} =$$

$$\begin{pmatrix} N & M^{(\sigma^{(1)})}e_{i_1}^T & M^{(\sigma^{(2)})}e_{i_2}^T & \dots & M^{(\sigma^{(k)})}e_{i_k}^T \\ 0 & \zeta_1 - f(w_1 \cdot \sigma^{(1)} \cdot y_{i_1}) & \dots & \dots & \dots \\ 0 & 0 & \zeta_2 - f(w_2 \cdot \sigma^{(2)} \cdot y_{i_2}) & \dots & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \zeta_k - f(w_k \cdot \sigma^{(k)} \cdot y_{i_k}) \end{pmatrix},$$

where

$$\zeta_j = \beta_0 \Lambda_0^{(w_j \cdot \sigma^{(j)})} N e_{i_j}.$$

Therefore the result follows. ■

7.2 The Complexity

In this subsection we analyze the complexity of the algorithm above. We will show that this algorithm achieves the lower bound of equivalence queries complexity for learning MA.

We show

Theorem 7. *Let $\mathcal{F}$ be a field, and $f : \Sigma^* \to \mathcal{F}$ be an MA of size s. Then f is learnable as MA from $r \le \lceil s/k \rceil$ equivalence queries and $O(r|\Sigma| \cdot msk + |\Sigma|^k s^2)$ membership queries in $O(r \cdot |\Sigma| ms^2 k + (s - r \cdot (k-1))|\Sigma|^k M(s))$ arithmetic operations, for some fixed integer k.*

Proof. We already proved that each time a counterexample is received we add k rows and columns to N. As a result, when the size of the target MA is s, r is bounded by $\lceil s/k \rceil$.

When the algorithm asks an equivalence query and gets a counterexample z, it finds the sets W and $\{\sigma^{(i)} | 1 \le i \le k\}$. Denote by $w_0 = z$, the algorithm finds a minimal length prefix w' of w_0 for which

$$(f(w' \cdot \sigma' \cdot y_1), f(w' \cdot \sigma' \cdot y_2), \dots, f(w' \cdot \sigma' \cdot y_\ell)) \neq \beta_0 \Lambda_0^{(w' \cdot \sigma')} N$$

for some $\sigma' \in \Sigma$.

Then it assigns $W \leftarrow \{w_1 = w'\}$ and $\sigma^{(1)} = \sigma'$. In the jth iteration, it finds a minimal prefix w' of w_{j-1} such that

$$(f(w' \cdot \sigma' \cdot \sigma^{(j-1)} \cdot y_1), f(w' \cdot \sigma' \cdot \sigma^{(j-1)} \cdot y_2), ..., f(w' \cdot \sigma' \cdot \sigma^{(j-1)} \cdot y_\ell)) \neq \beta_0 \Lambda_0^{(w' \cdot \sigma' \cdot \sigma^{(j-1)})} N$$

for some $\sigma' \in \Sigma$.

Then $W \leftarrow W \cup \{w_j = w'\}$ and $\sigma^{(j)} = \sigma' \cdot \sigma^{(j-1)}$. The algorithm runs k iterations, each iteration j takes at most $|\Sigma| ms$ membership queries and computes $\beta_0 \Lambda_0^{(w' \cdot \sigma' \cdot \sigma^{(j-1)})} N$ for every prefix w' of w_{j-1} which takes $O(|\Sigma| ms^2)$ arithmetic operations. Thus after each counterexample, the total number of membership queries asked is $O(|\Sigma| msk)$ and the total number of arithmetic operations is $O(|\Sigma| ms^2 k)$. Notice that, we already computed $\Lambda_0^{\sigma^{(j-1)}} N$ since $|\sigma^{(j-1)}| \le k$.

Now, we have $\hat{X} = W \bigcup X$ and $\hat{Y} = Y \bigcup \{\sigma^{(j)} \cdot y_{i_j} | j = 1 \dots, k\}$ and i_j is any entry that satisfies (6).

All entries of $\hat{N}$ are known. To update the matrices $\hat{M}^{(\tau)}$ for all $\tau \in \Sigma^{\leq k}$, the algorithm asks $O(s^2 \cdot |\Sigma|^k)$ membership queries during its run.

By Lemma 5 below we can find N^{-1} and $\Lambda_0^{(\sigma)}$, for every $\sigma \in \Sigma$, in $O(s^2 \log(s)|\Sigma|)$ arithmetic operations.

Finally our algorithm performs the internal checking step, it needs to compute $\Lambda_0^{(\tau)} N$ for every $\tau \in \Sigma^{\leq k}$. For this, the algorithm multiplies $\sum_{i=1}^{k} |\Sigma|^i = O(|\Sigma|^k)$ matrices each of size at most $s \times s$. Since multiplying two matrices of size $s \times s$ takes $M(s)$ arithmetic operations, to perform internal checking the algorithm needs $O(|\Sigma|^k M(s))$ arithmetic operations.

When finding a counterexample z during the internal checking, z will be of the form $x_i \cdot \tau' \cdot y_j$ for some $x_i \in X$, $y_j \in Y$ and $\tau' \in \Sigma^{\leq k}$. For minimal length $\tau' = \tau'_1 \tau'_2 \cdots \tau'_{|\tau'|}$, such that z remains a counterexample, the algorithm adds $x_i \cdot \tau'_1 \tau'_2 \ldots \tau'_{|\tau'|-1}$ to X and $\tau'_{|\tau'|} \cdot y_j$ to Y.

In this case, updating N and $\Lambda^{(\sigma)}$, for every $\sigma \in \Sigma$, will be as in the proof of Theorem 3 and it will cost $O(|\Sigma|s^2)$ arithmetic operations at most.

To conclude, if the algorithm asks r equivalence queries, each time it asks $O(|\Sigma|msk)$ membership queries and needs $O(|\Sigma|ms^2k)$ arithmetic operations. Consequently, the algorithm will find $s - r \cdot k$ counterexamples, by asking membership queries in the internal check, each time it needs $O(|\Sigma|s^2)$ arithmetic operations. Moreover, each time the algorithm performs an internal checking it takes $O(|\Sigma|^k M(s))$ arithmetic operations.

Summing all the above, when learning a target MA function of size s, we need $O(r \cdot |\Sigma|msk + |\Sigma|^k s^2)$ membership queries and $O(r \cdot |\Sigma|ms^2k + (s - r \cdot (k - 1))|\Sigma|^k M(s))$ arithmetic operations. ■

Lemma 5. *Let N and M be two $\ell \times \ell$ matrices with entries from $\mathcal{F}$ and $\Lambda = MN^{-1}$. Let $u, \Delta \in \mathcal{F}^{\log \ell \times \ell}$ be such that $u = \Delta N$.*

$$\hat{N} = \begin{pmatrix} N & v^T \\ u & \xi \end{pmatrix}, \hat{M} = \begin{pmatrix} M & p^T \\ q & \eta \end{pmatrix}$$

where $\hat{N}$ is nonsingular matrix where $v, q, p \in \mathcal{F}^{\log \ell \times \ell}$ and $\xi, \eta \in \mathcal{F}^{\log \ell \times \ell}$. Then

$$\hat{N}^{-1} = \begin{pmatrix} N^{-1} - (N^{-1}v^T\omega^{-1})\Delta & N^{-1}v^T\omega^{-1} \\ \omega^{-1}\Delta & -\omega^{-1} \end{pmatrix}$$

where $\omega = \lambda v^T - \xi$ and

$$\hat{M}\hat{N}^{-1} = \begin{pmatrix} \Lambda - (\Lambda v^T\omega^{-1})\Delta + p^T\omega^{-1}\Delta & \Lambda v^T\omega^{-1} - p^T\omega^{-1} \\ (qN^{-1}) - ((qN^{-1})v^T\omega^{-1})\Delta + \eta\omega^{-1}\Delta & (qN^{-1})v^T\omega^{-1} - \eta\omega^{-1} \end{pmatrix}.$$

See the full paper for the MAF results.

Acknowledgement. We would like to thank Lawrance Khoury for his contribution to the preliminary version of the paper.

References

1. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2), 87-106, 1987.
2. D. Angluin. Queries and concept learning. *Machine Learning*, 2, 319–342, 1987.
3. M. Bläser. Lower bounds for the multiplicative complexity of matrix multiplication. *Comput. Complexity,* 8(3), 203–226, 1999.
4. A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, S. Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3), 506–530, 2000.
5. F. Bergadano, N. H. Bshouty, C. Tamon, S. Varricchio. On Learning Programs and Small Depth Circuits. *EuroCOLT*. 150–161, 1997.
6. F. Bergadano, N. H. Bshouty, S. Varricchio. Learning Multivariate Polynomials from Substitution and Equivalence Queries. http://www.eccc.uni-trier.de/eccc , 1996.
7. F. Bergadano, D. Catalano, S. Varricchio. Learning Sat-k-DNF Formulas from Membership Queries.*STOC 96*, 126–130, 1996.
8. J. L. Balcázar, J. Díaz, R. Gavaldá, O. Watanabe. A note on the query complexity of learning DFA. *ALT 92*. 53–62. 1992.
9. N. H. Bshouty, S. A. Goldman, T. R. Hancock, S. Matar. Asking Questions to Minimize Errors. *J. Comput. Syst. Sci.* 52(2), 268–286, 1996.
10. A. Beimel, E. Kushilevitz: Learning Boxes in High Dimension. *Algorithmica*, 22(1/2), 76–90, 1998.
11. N. H. Bshouty, C. Tamon, D. K. Wilson. Learning matrix functions over rings. *Algorithmica* 22(1/2), 91–111, 1998.
12. F. Bergadano, S. Varricchio. Learning Behaviors of Automata from Multiplicity and Equivalence Queries. *SIAM J. Comput.* 25(6), 1268–1280, 1996.
13. F. Bergadano and S. Varricchio. Learning behaviors of automata from shortest counterexamples. *EuroCOLT 95*, 380-391, 1996.
14. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9, 251-280, 1990.
15. N. Elkies. On finite sequences satisfying linear recursions. Available at http://xxx.lanl.gov/abs/math.CO/0105007.
16. M. J. Kearns, L. G. Valiant. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata. *J. ACM*, 41(1), 67–95, 1994.
17. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2, 285–318, 1987.
18. N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2), 212–261, 1994.
19. W. Maass and G. Turán. Lower bound methods and seperation results for on-line learning models. *Machine Learning* , 9, 107–145, 1992.
20. H. Ohnishi, H. Seki, and T. Kasami. A polynomial time learning algorithm for recognizable series.*IEICE Transactions on Information and Systems*, E77-D(5), 1077–1085, 1994.
21. A. Shpilka. Lower Bounds for Matrix Product. *SIAM J. Comput.* 32(5), 1185–1200, 2003.