SIAM J. COMPUT.
Vol. 25, No. 6, pp. 1268–1280, December 1996

007

LEARNING BEHAVIORS OF AUTOMATA FROM MULTIPLICITY AND EQUIVALENCE QUERIES*

FRANCESCO BERGADANO† AND STEFANO VARRICCHIO‡

Abstract. We consider the problem of identifying the behavior of an unknown automaton with multiplicity in the field Q of rational numbers (Q-automaton) from multiplicity and equivalence queries. We provide an algorithm which is polynomial in the size of the Q-automaton and in the maximum length of the given counterexamples. As a consequence, we have that Q-automata are probably approximately correctly learnable (PAC-learnable) in polynomial time when multiplicity queries are allowed. A corollary of this result is that regular languages are polynomially predictable using membership queries with respect to the representation of unambiguous nondeterministic automata. This is important since there are unambiguous automata such that the equivalent deterministic automaton has an exponentially larger number of states.

Key words. PAC-learning, exact ientification, learning from examples, learning from queries, equivalence queries, multiplicity queries, membership queries, multiplicity automata, probabilistic automata, unambiguous nondeterministic automata

AMS subject classifications. 68Q68, 68Q70, 68Q75, 68T05

1. Introduction.

Learning automata from examples and from queries has been extensively investigated in the past, and important results have been obtained recently. Early on, it was noticed that the problem of exactly identifying a minimum automaton consistent with given data is NP-complete [10]. Similar results may be proved for regular expressions [1]. Even simply approximating the minimum consistent deterministic finite automaton (DFA) problem is not feasible [14]. Gold [10] proves that polynomial identification in the limit is still possible in the sense that an inductive inference machine will take polynomial time when processing a new example. However, this may seem unsatisfactory since the number of examples needed may be arbitrarily large. A natural direction generally followed in machine learning (see Chapter 2.3 in [6] and references therein) was to consider a learner who did not just passively receive data but who was able to ask queries.

Some questions—called *membership queries*—may consist of asking an oracle whether a particular string belongs to the target language. Angluin [2] proved that if we start from a set of strings that lead to every reachable state in the target automaton, a polynomial number of membership queries is sufficient for exact identification. However, if such a set of strings is not available, even if we know the size n of the target automaton, the number of queries needed is exponential in n.

Another possibility is found in *equivalence queries*: asking an oracle whether a guess is correct and obtaining a counterexample if it is not. We shall also assume that the counterexamples have a maximum length m. It may be shown [5] that there is no polynomial-time algorithm to exactly identify automata from equivalence queries only. However, there is a polynomial algorithm if both equivalence and membership queries are used [3].

* Received by the editors November 1, 1993; accepted for publication (in revised form) February 14, 1995.

† Dimartimento di Informatica, Università di Torino, 185 Corso Svizzera, 10149 Torino, Italy (bergadan@di.unito.it). The research of this author was supported by the European Union under IV Framework ESPRIT, Long Term Research Project "ILP2" (contract 20237).

‡ Dipartimento di Matematica Pura ed Applicata, Università dell'Aquila, Via Vetoio, 67010 L'Aquila, Italy (varricch@univaq.it). The research of this author was partly supported by the Italian Minister of Universities and by ESPRIT–EBRA project ASMICS (contract 6317).

Equivalence and membership queries then seem to be a necessary requirement for learning deterministic finite-state automata. It remains to be seen if stronger formalisms may be learned under the same framework. Following preliminary results reported in [7], this paper gives a positive answer in the direction of behaviors of nondeterministic finite-state automata, i.e., functions that assign to every string the number of its accepting paths in a nondeterministic finite-state acceptor. Such functions, as defined in the next section, will be described in the more general framework of automata with multiplicity.

We introduce the notion of a *multiplicity query*. In the case of a nondeterministic automaton, a multiplicity query returns the number of accepting paths for a given string. We show that behaviors of automata with multiplicity may be identified in polynomial time with multiplicity and equivalence queries. This implies that they are probably approximately correctly learnable (PAC-learnable) with multiplicity queries. If we restrict the result to unambiguous nondeterministic automata, multiplicity queries must return either 0 or 1 and reduce to membership queries. As a consequence of our main result, we may then PAC-learn with membership queries a representation of a regular language L in polynomial time with respect to the size of an unambiguous nondeterministic automaton that accepts L. This is an improvement over the result of Angluin [3] because there are unambiguous automata such that the equivalent DFA has an exponentially larger number of states [16]. However, it must be noted that the learned representation of the regular language is not an unambiguous nondeterministic automaton. Therefore, unambiguous nondeterministic finite automata (NFAs) are only shown to be PAC-predictable with membership queries. Our main result also applies to probabilistic automata. In that case, the multiplicity of a given string represents the probability of accepting that string. Again, our main result implies the PAC-predictability of probabilistic automata with multiplicity queries of this kind. This is an improvement over the results of Tzeng [17], where stronger queries are needed, giving the probability of reaching a given state with a given string.

2. Multiplicity automata.

Automata with multiplicity, also called multiplicity automata, are the most important generalizations of classical automata theory. In recent years, their significant development has helped in solving old problems in automata theory. In [11], using multiplicity automata, the decidability of the equivalence problem for deterministic multitape automata has been solved; in [18], a similar result has been shown for unambiguous regular languages in a free partially commutative monoid.

Let M be an NFA. We can consider the so-called *behavior* of M, which is the map that associates with any word the number of its different accepting paths. More generally, we can assign a multiplicity to the initial states, the final states, and the edges of the automaton so that the corresponding behavior must take into account the assigned multiplicities. In this way, we can construct a theory which is general enough to contain classical and probabilistic automata as particular objects. Multiplicity automata have been extensively studied in theoretical computer science, and we refer to [8], [9], and [15]; here we recall some notation and definitions.

Let K be a field and A^* be the free monoid over a finite alphabet A; we consider the set K^{A^*} of all the applications $S : A^* \to K$. An element $S \in K^{A^*}$ will be called a *K-subset* of A^* or simply a *K-set*. Following the standard notation on K-sets, for any $S \in K^{A^*}$ and $u \in A^*$, we will denote $S(u)$ by (S, u). In what follows, we shall consider $\mathbb{Q}$-sets, where $\mathbb{Q}$ denotes the field of rational numbers.

We denote by $\mathbb{Q}^{n\times n}$ the set of all square $n \times n$ matrices with entries in $\mathbb{Q}$. We shall consider $\mathbb{Q}^{n\times n}$ to be equipped with the row-by-column product; the identity matrix is denoted by Id. A map $\mu : A^* \to \mathbb{Q}^{n\times n}$ is a morphism if $\mu(\epsilon) = \mathrm{Id}$ and for any $w \in A^+$, $w = a_1a_2\cdots a_n$, $a_i \in A$, we have $\mu(w) = \mu(a_1)\mu(a_2)\cdots\mu(a_n)$. A $\mathbb{Q}$-set $S \in \mathbb{Q}^{A^*}$ is called *recognizable* or *representable* if there exists a positive integer n, λ, $\gamma \in \mathbb{Q}^n$ and a morphism $\mu : A^* \to \mathbb{Q}^{n\times n}$ such that for any $w \in A^*$

$$(S, w) = \lambda\mu(w)\gamma,$$

where λ and γ are to be considered row and column vectors, respectively. The triple (λ, μ, γ) is called a *linear representation* of S of dimension n or a $\mathbb{Q}$-*automaton* for S.

A *nondeterministic automaton* is a 5-tuple $M = (A, Q, E, I, F)$, where A is the input alphabet, Q is a finite set of *states*, $E \subseteq Q\times A\times Q$ is a set of *edges*, and $I, F \subseteq Q$ are, respectively, the sets of the *initial* and *final* states. Let $w = a_1a_2\cdots a_n \in A^*$. An *accepting path* for w is any sequence $\pi = (p_1, a_1, p_2)(p_2, a_2, p_3)\cdots(p_n, a_n, p_{n+1})$ with $p_1 \in I$, $p_{n+1} \in F$ and $(p_i, a_i, p_{i+1}) \in E$ for $1 \le i \le n$. The language accepted by M is the set $L(M)$ of all the words which have at least one accepting path; M is *unambiguous* if for any word $w \in L(M)$, there exists only one accepting path for w. We can associate with M a $\mathbb{Q}$-set $S_M \in \mathbb{Q}^{A^*}$, also called the *behavior* of M, defined as follows: for any $w \in A^*$, (S_M, w) is the number of different paths which are accepting for w. Let $Q = 1, 2, \dots, n$ and let λ, $\gamma \in \mathbb{Q}^n$ be the characteristic vectors, respectively, of I and F; consider the morphism $\mu : A^* \to \mathbb{Q}^{n\times n}$, defined by $\mu(a)_{ij} = 1$ if $(i, a, j) \in E$ and $\mu(a)_{ij} = 0$ otherwise. Then we can easily prove (cf. [9, p. 137]) that for any $w \in A^*$

$$(S_M, w) = \lambda\mu(w)\gamma.$$

In particular, S_M is representable and, when M is unambiguous, S_M corresponds to the characteristic function of $L(M)$.

In general, a linear representation (λ, μ, γ) of dimension n can be regarded as an "automaton" whose set of states is $Q = \{1, 2, \dots n\}$; initial and final states are defined as $\mathbb{Q}$-subsets of Q, while edges are a $\mathbb{Q}$-subset of $Q \times A \times Q$. Indeed, λ_i (resp. γ_i) represents the multiplicity of i as an initial state (resp. final state) and $\mu(a)_{i,j}$ represents the multiplicity of the edge (i, a, j). Probabilistic automata are particular $\mathbb{Q}$-automata [13]. A probabilistic automaton P can be represented by means of a linear representation (λ, μ, γ) with the following constraints: $\sum_{i=1}^{n} \lambda_i = 1$ and $\sum_{j=1}^{n} \mu(a)_{i,j} = 1$ for any $a \in A$ and $i \in \{1, 2, \dots, n\}$; moreover, $0 \le \lambda_i \le 1$, $\gamma_i \in \{0, 1\}$, and $0 \le \mu(a)_{i,j} \le 1$ for any $a \in A$ and $i, j \in \{1, 2, \dots, n\}$. Informally, λ_i represents the probability of i being an initial state, γ_i is 1 iff i is an accepting state, and $\mu(a)_{i,j}$ represents the probability of arriving in state j, starting from state i, and reading the input symbol a. Then the probability that P accepts when started with the distribution probability λ on Q and reading w is exactly $\lambda\mu(w)\gamma$. Finally, we shall need the following definitions.

DEFINITION 2.1. *For any string $u \in A^*$ and a $\mathbb{Q}$-set S, the $\mathbb{Q}$-set S_u is defined by* $(\forall w \in A^*)\,(S_u, w) = (S, uw)$.

DEFINITION 2.2. *For any set of strings $E \subseteq A^*$,*

$$S \equiv_E T \quad \text{iff } (\forall w \in E)\,(S, w) = (T, w).$$
$$S \equiv T \text{ stands for } S \equiv_{A^*} T.$$

We shall use an oracle for answering *multiplicity queries* for any string w, i.e., for providing the value of (S, w), where S is the target $\mathbb{Q}$-set.

3. Observation tables. Based on previous work by Angluin on deterministic finite-state automata [3], we now introduce the concept of an observation table for a $\mathbb{Q}$-set S.

DEFINITION 3.1. *Let $S \in \mathbb{Q}^{A^*}$; an observation table is a triple $\mathcal{T} = (P, E, T)$, where P and E are sets of strings, P is prefix-closed, E is suffix-closed, and $T : (P \cup PA)E \to \mathbb{Q}$ gives observed values of S, i.e., for all strings $w \in (P \cup PA)E, T(w) = (S, w)$.*

Consequently, an observation table provides particular values for the target $\mathbb{Q}$-set S. Such values are obtained by means of multiplicity queries once the sets P and E are fixed. In Angluin's method for the exact identification of regular languages, the set P corresponds to states in an accepting DFA and the table contains some of the transitions. Here P determines a set $\{S_u | u \in P\}$ that will be useful in defining the target $\mathbb{Q}$-set S via linear dependencies. We then have the corresponding notions of closed and consistent observation tables.

DEFINITION 3.2. *An observation table (P, E, T) is closed iff $\forall u \in P$, $\forall a \in A$, there exists a coefficient $\alpha_v \in \mathbb{Q}$ for each $v \in P$ such that*

$$S_{ua} \equiv_E \sum_{v \in P} \alpha_v S_v. \tag{1}$$

DEFINITION 3.3. *An observation table (P, E, T) is consistent iff for any choice of coefficients $\beta_v \in \mathbb{Q}$ for each $v \in P$,*

$$\sum_{v \in P} \beta_v S_v \equiv_E 0 \Rightarrow (\forall a \in A) \sum_{v \in P} \beta_v S_{va} \equiv_E 0. \tag{2}$$

In Angluin's work (see also [2]), a natural notion of completeness was defined for P that required that all states in the target DFA have a representative in P. Here we have an analogous notion that requires that $\{S_u | u \in P\}$ be sufficient to establish all of the linear dependencies needed.

DEFINITION 3.4. *P is a complete set of strings for S iff $\forall u \in P$, $\forall a \in A$, there exists a coefficient $\lambda_v \in \mathbb{Q}$ for each $v \in P$ such that*

$$S_{ua} \equiv \sum_{v \in P} \lambda_v S_v. \tag{3}$$

When a table (P, E, T) is consistent and P is complete, the linear dependencies that are observed on E are valid for any string in A^*, as proved in the following.

THEOREM 3.5. *Let (P, E, T) be a consistent observation table, where P is a complete set of strings for S; then*

$$\sum_{v \in P} \beta_v S_v \equiv_E 0 \Rightarrow \sum_{v \in P} \beta_v S_v \equiv 0. \tag{4}$$

Proof. We shall prove the theorem by induction on $|y|$ by showing that for any $y \in A^*$,

$$\sum_{v \in P} \beta_v S_v \equiv_E 0 \Rightarrow \sum_{v \in P} \beta_v S_{vy} \equiv_E 0. \tag{5}$$

Base. $y = \epsilon$ and the thesis is trivially true.

Inductive step. Let $y = wb$ with $b \in A$.

By using the completeness of P a sufficient number of times, given $v \in P$, we may find coefficients $\lambda_{u,v}$, $u \in P$, such that

$$S_{vw} \equiv \sum_{u \in P} \lambda_{u,v} S_u. \tag{6}$$

Then for $x \in E$,

$$\begin{aligned} \sum_{v \in P} \beta_v (S_{vwb}, x) &= \sum_{v \in P} \beta_v (S_{vw}, bx) = \sum_{v \in P} \beta_v \sum_{u \in P} \lambda_{u,v} (S_u, bx) \\ &= \sum_{u \in P} \sum_{v \in P} \beta_v \lambda_{u,v} (S_u, bx) = \sum_{u \in P} \gamma_u (S_u, bx), \\ \text{where } \gamma_u &= \sum_{v \in P} \beta_v \lambda_{u,v}. \end{aligned} \tag{7}$$

By the inductive hypothesis, $\sum_{v \in P} \beta_v S_{vw} \equiv_E 0$; then, using (6),

$$\sum_{v \in P} \beta_v \sum_{u \in P} \lambda_{u,v} S_u \equiv \sum_{u \in P} \gamma_u S_u \equiv_E 0.$$

Again using the consistency of the table, we have

$$\sum_{u \in P} \gamma_u S_{ub} \equiv_E 0, \tag{8}$$

and, using this in (7), $\sum_{v \in P} \beta_v S_{vwb} \equiv_E 0$.

This completes the proof of (5). The theorem then follows from the fact that, since $\epsilon \in E$,

$$\left[(\forall y \in A^*) \sum_{v \in P} \beta_v S_{vy} \equiv_E 0 \right] \Rightarrow \sum_{v \in P} \beta_v S_v \equiv 0. \qquad \square \tag{9}$$

Consequently, the linear dependencies that show the table is closed are also valid in A^*.

COROLLARY 3.6. *Let (P, E, T) be a consistent observation table, where P is a complete set of strings for S; then for any $a \in A$,*

$$S_{ua} \equiv_E \sum_{v \in P} \lambda_v S_v \Rightarrow S_{ua} \equiv \sum_{v \in P} \lambda_v S_v. \tag{10}$$

Proof. Since P is complete, $S_{ua} \equiv \sum_{v \in P} \lambda'_v S_v$ for some $\lambda'_v \in \mathbb{Q}$. Therefore, if $S_{ua} \equiv_E \sum_{v \in P} \lambda_v S_v$, then $\sum_{v \in P} (\lambda'_v - \lambda_v) S_v \equiv_E 0$. By Theorem 3.5, $\sum_{v \in P} (\lambda'_v - \lambda_v) S_v \equiv 0$, and

$$S_{ua} \equiv \sum_{v \in P} \lambda'_v S_v \equiv \sum_{v \in P} \lambda_v S_v. \qquad \square$$

4. The learning algorithm. As is explained in what follows, there are stages in which the learning process builds a consistent and closed table. Here we only want to show how from such a table (P, E, T), we can guess a $\mathbb{Q}$-set $M(P, E, T)$ by basing its representation upon the existing linear dependencies:

- Let $P = \{u_1, \ldots, u_k\}$, with $u_1 = \epsilon$.

- For all $a \in A$, compute $\hat{\mu}(a)$ that satisfies

$$S_{u_i a} \equiv_E \sum_j \hat{\mu}(a)_{u_i,u_j} S_{u_j}. \tag{11}$$

Such a matrix exists because the table is closed. Moreover, the values of $\hat{\mu}(a)_{u_i,u_j}$ can be computed by solving the system of linear equations $(S_{u_i a}, v) = \sum_j x_{i,j}(S_{u_j}, v)$ with $v \in E$ in the unknowns $x_{i,j}$.

- Let $\hat{\lambda} = (1, 0, \ldots, 0)$ and $\hat{\gamma} = ((S_{u_1}, \epsilon), (S_{u_2}, \epsilon), \ldots, (S_{u_k}, \epsilon))$. The value of (S_{u_j}, ϵ) is found in the table since $u_j \in P$ and $\epsilon \in E$. Obviously, $\hat{\mu}(a)_{u_i,u_j}$ is the value at row i and column j of the matrix $\hat{\mu}(a)$. Let $\hat{\mu}(a_1 a_2 \cdots a_r) = \hat{\mu}(a_1)\hat{\mu}(a_2)\cdots\hat{\mu}(a_r)$, $a_i \in A$. Define the constructed $\mathbb{Q}$-set M with $(M, w) = \hat{\lambda}\hat{\mu}(w)\hat{\gamma}$.

THEOREM 4.1. *If P is a complete set of strings for S and (P, E, T) is a closed and consistent table, then $M(P, E, T) \equiv S$.*

Proof. Again, let $P = \{u_1, \ldots, u_k\}$ with $u_1 = \epsilon$. Since the table is closed, for any $a \in A$ $S_{u_i a} \equiv_E \sum_j \hat{\mu}(a)_{u_i,u_j} S_{u_j}$ and by Corollary 3.6, $S_{u_i a} \equiv \sum_j \hat{\mu}(a)_{u_i,u_j} S_{u_j}$. This may be easily generalized to any string t in A^*; by induction on $|t|$, we derive $S_{u_i t} \equiv \sum_j \hat{\mu}(t)_{u_i,u_j} S_{u_j}$. In fact let $t = sb$ with $b \in A$. By the induction hypothesis, we have $S_{u_i s} \equiv \sum_k \hat{\mu}(s)_{u_i,u_k} S_{u_k}$, which implies that $S_{u_i sa} \equiv \sum_k \hat{\mu}(s)_{u_i,u_k} S_{u_k a}$. On the other hand, from $S_{u_k a} \equiv \sum_j \hat{\mu}(a)_{u_k,u_j} S_{u_j}$, we derive $S_{u_i sa} \equiv \sum_k \sum_j \hat{\mu}(s)_{u_i,u_k} \hat{\mu}(a)_{u_k,u_j} S_{u_j} = \sum_j \hat{\mu}(sa)_{u_i,u_j} S_{u_j}$. Then

$$(S_{u_i}, t) = (S_{u_i t}, \epsilon) = \sum_j \hat{\mu}(t)_{u_i,u_j} (S_{u_j}, \epsilon). \tag{12}$$

Since $u_1 = \epsilon$, we have

$$(S, t) = \sum_j \hat{\mu}(t)_{u_1,u_j} (S_{u_j}, \epsilon) = \hat{\lambda}\hat{\mu}(t)\hat{\gamma} = (M, t). \qquad \square$$

We are now left with three problems: (i) closing a table; (ii) making a table consistent; (iii) making P complete. However, we will obtain completeness only indirectly and will return to it later.

4.1. Closing a table.

Given a table (P, E, T) and $u \in P$, suppose that S_{ua} is linearly independent of $\{S_v | v \in P\}$ with respect to E in the sense that there are no coefficients $\lambda_{u,v}$ such that $S_{ua} \equiv_E \sum_{v \in P} \lambda_{u,v} S_v$. In this case, ua is added to P, and the table is again checked for closure.

This procedure must terminate; more precisely, if the correct $\mathbb{Q}$-set S is representable with $(S, x) = \lambda\mu(x)\gamma$, where $\lambda, \gamma \in \mathbb{Q}^n$ and $\mu : A^* \to \mathbb{Q}^{n \times n}$ is a morphism, then at most n strings can be added to P when closing the table.

In fact, it should be noted that when ua is added to P as indicated above, the dimension of $\{\lambda\mu(v) | v \in P\}$ as a subset of the vector space $\mathbb{Q}^n$ is increased by one. Otherwise, $\lambda\mu(ua)$ would be equal to $\sum_{v \in P} \beta_v \lambda\mu(v)$ for some coefficients β_v and

$$(S_{ua}, x) = (S, uax) = \lambda\mu(ua)\mu(x)\gamma = \sum \beta_v \lambda\mu(v)\mu(x)\gamma = \sum \beta_v (S_v, x),$$

i.e., S_{ua} would depend linearly on $\{S_v | v \in P\}$. Since the dimension of $\{\lambda\mu(v) | v \in P\}$ is at most n, we cannot close the table more than n times. The above discussion does not depend on E.

4.2. Making tables consistent. Given a table (P, E, T) and a symbol $a \in A$, consider the systems of linear equations

$$\text{(a)} \sum_{v \in P} \beta_v S_v \equiv_E 0, \qquad \text{(b)} \sum_{v \in P} \beta_v S_{va} \equiv_E 0$$

with β_v as unknowns. Check if every solution of system (a) is also a solution of system (b). In this case, the table is consistent. Otherwise, let β'_v, $v \in P$, be some solutions of (a) that are not solutions of (b) and let $x \in E$ such that $\sum_{v \in P} \beta_v (S_{va}, x) \neq 0$. Add ax to E. A method for checking whether every solution of (a) is also a solution of (b) is outlined in §4.4.

Suppose that S has a linear representation (λ, μ, γ) of dimension n; there cannot be more than n such additions to E because every time a new string ax is added, the dimension of $\{\mu(w)\gamma | w \in E\}$ is increased by one. In fact, if $\mu(ax)\gamma = \sum_{w \in E} \delta_w \mu(w)\gamma$, then

$$\begin{aligned}
\sum_{v \in P} \beta_v (S_{va}, x) &= \sum_{v \in P} \beta_v \lambda \mu(v) \mu(ax) \gamma \\
&= \sum_{v \in P} \beta_v \lambda \mu(v) \sum_{w \in E} \delta_w \mu(w) \gamma \\
&= \sum_{w \in E} \delta_w \sum_{v \in P} \beta_v \lambda \mu(vw) \gamma \\
&= \sum_{w \in E} \delta_w \sum_{v \in P} \beta_v (S_v, w) = 0,
\end{aligned}$$

i.e., ax would not have been added to E.

4.3. The algorithm. We may now describe the procedure for exactly identifying S from multiplicity queries and counterexamples.

```
$\mathcal{T} \leftarrow (\{\epsilon\},\{\epsilon\},T)$, where $(T,\epsilon) = (S,\epsilon)$.
Repeat
    • make the table closed and consistent ($P$ and $E$ are extended and the entries of $T$ are filled in by multiplicity queries); while closing the table, the hypothesized $\mathbb{Q}$-set $M$ is obtained;
    • ask for a counterexample $t$ to $M(P, E, T)$ by means of an equivalence query;
    • add $t$ and its prefixes to $P$
until $M$ is correct.
```

The main loop makes the table closed and consistent as described in §§4.1 and 4.2 and then constructs a guess M which is based on the observed linear dependencies. We shall now prove that if S has a linear representation (λ, μ, γ) of dimension n, after at most n equivalence queries, we will have a correct guess, i.e., $M \equiv S$. We need the following result.

LEMMA 4.2. *Let $u \in P$ and $t \in uA^*$, and suppose that for every $x \in A^*$ such that $t = uxz$, S_{ux} depends linearly on $\{S_v | v \in P\}$. Then for every prefix ux of t, after the table has been made closed and consistent during the execution of the algorithm, we have*

$$S_{ux} \equiv_E \sum_{v \in P} \hat{\mu}(x)_{u,v} S_v, \tag{13}$$

where $\hat{\mu} : A^* \to \mathbb{Q}^{k \times k}$ *is the morphism that corresponds to the observed linear dependencies as computed in* (11), *which is obtained when closing the table.*

Proof (by induction on $|x|$).

Base. $x = \epsilon$ and $S_u \equiv_E \sum_{v \in P} \hat{\mu}(\epsilon)_{u,v} S_v$ since $\hat{\mu}(\epsilon)$ is the identity matrix.

Inductive step. Let $x = yb$ and assume that $S_{uy} \equiv_E \sum_{v \in P} \hat{\mu}(y)_{u,v} S_v$.

Since uy is a prefix of t, S_{uy} must depend linearly on $\{S_v | v \in P\}$, i.e.,

$$S_{uy} \equiv \sum_{v \in P} \alpha_v S_v. \tag{14}$$

This along with the inductive hypothesis gives

$$\sum_{v \in P} (\alpha_v - \hat{\mu}(y)_{u,v}) S_v \equiv_E 0. \tag{15}$$

Then, since the table is consistent,

$$\sum_{v \in P} (\alpha_v - \hat{\mu}(y)_{u,v}) S_{vb} \equiv_E 0, \quad \text{i.e.,} \quad \sum_{v \in P} \alpha_v S_{vb} \equiv_E \sum_{v \in P} \hat{\mu}(y)_{u,v} S_{vb}. \tag{16}$$

Using (14) again, we have

$$S_{uyb} \equiv_E \sum_{v \in P} \hat{\mu}(y)_{u,v} S_{vb}. \tag{17}$$

Since $v \in P$, we may substitute $S_{vb} \equiv \sum_{w \in P} \hat{\mu}(b)_{v,w} S_w$ in the previous equation, which yields

$$S_{ux} \equiv S_{uyb} \equiv_E \sum_{v} \hat{\mu}(y)_{u,v} \sum_{w} \hat{\mu}(b)_{v,w} S_w \equiv \sum_{w} \hat{\mu}(yb)_{u,w} S_w. \qquad \square$$

THEOREM 4.3. *Let* $(S,t) \neq (M,t)$. *Then there is a prefix* t_0 *of* t *such that* S_{t_0} *is linearly independent of* $\{S_v | v \in P\}$.

Proof. If all prefixes t_0 of t would make S_{t_0} depend linearly on $\{S_v | v \in P\}$, then, by setting $u = \epsilon$ in the previous lemma and because $\epsilon \in E$, $(S,t) = (S_{\epsilon t}, \epsilon) = \sum_{v \in P} \hat{\mu}(t)_{\epsilon,v} (S_v, \epsilon) = \hat{\lambda} \hat{\mu}(t) \hat{\gamma}$. Then, however, $(M,t) = \hat{\lambda} \hat{\mu}(t) \hat{\gamma}$ would not be different from (S,t). □

COROLLARY 4.4. *If* S *has a linear representation of dimension* n, *then after at most* n *iterations, the algorithm stops.*

Proof. Again suppose that the correct $\mathbb{Q}$-set S is representable with $(S,x) = \lambda \mu(x) \gamma$, where $\lambda, \gamma \in \mathbb{Q}^n$ and $\mu : A^* \to \mathbb{Q}^{n \times n}$. The algorithm will ask for a counterexample t and will add t and all of its prefixes to P. By Theorem 4.3, at each iteration, some prefix t_0 of the counterexample t is such that S_{t_0} is linearly independent of $\{S_u | u \in P\}$. However, $(S_{t_0}, x) = (S, t_0 x) = \lambda \mu(t_0) \mu(x) \gamma$ and for $u \in P$, $(S_u, x) = (S, ux) = \lambda \mu(u) \mu(x) \gamma$. Consequently, the dimension of $\{\lambda \mu(u) | u \in P\}$ is increased by one when t_0 is added to P. In fact, if by way of contradiction, $\lambda \mu(t_0) = \sum_{u \in P} \delta_u \lambda \mu(u)$ before t and its prefixes were added to P, then for all $x \in A^*$, $(S_{t_0}, x) = \lambda \mu(t_0) \mu(x) \gamma = \sum_{u \in P} \delta_u \lambda \mu(u) \mu(x) \gamma = \sum_{u \in P} \delta_u (S_u, x)$, i.e., S_{t_0} would depend linearly on $\{S_u | u \in P\}$. However, the dimension of $\{\lambda \mu(u) | u \in P\}$ cannot be larger than n. □

4.4. Complexity analysis. All we need to do is reorganize some of the previous results and determine the complexity of computing the linear dependencies. Let n be the dimension of a linear representation of S. We have the following:

- The main loop in the algorithm is repeated at most n times (Corollary 4.4).
- $|\mathrm{E}| \leq n$ (§4.2).
- For the cardinality of P, the discussion is slightly more involved. From the discussions of §§4.1 and 4.3, we see that every time either (i) a string is added to P while closing the table or (ii) a counterexample is processed, the dimension of $\{\lambda\mu(v)|v \in P\}$ is increased by at least one. The worst case is when this always happens with the counterexamples and the main loop in the algorithm is repeated exactly n times, because the prefixes of the counterexamples also need to be added to P. If m is the maximum length of a counterexample, then $|P| \leq nm$.
- For every $a \in A$ and for every $u \in P$, the table needs to be closed. This amounts to solving at most knm systems of $|E|$ equations in $|P|$ unknowns—in the worst case, n simultaneous equations in nm unknowns. This can be done with Gauss's method with complexity $O(n^3m)$. Consequently, the complexity of closing the table is $O(kn^4m^2)$. We assume that the entries of the table are rational numbers of limited size; otherwise, the basic arithmetic operations involved in solving the systems of equations would be of arbitrary complexity. In general, it will be sufficient to require the entries of the table to be polynomial in n.
- Checking for consistency was described in §4.2 and requires the algorithm to confront the two systems of equations

$$\text{(a)} \sum_{v \in P} \beta_v S_v \equiv_E 0, \qquad \text{(b)} \sum_{v \in P} \beta_v S_{va} \equiv_E 0$$

with β_v as unknowns. The table is consistent if every solution of (a) is also a solution of (b). This is the same as checking whether the $|E|$ equations of system (b) do not add additional constraints, i.e., if the corresponding vectors of coefficients depend linearly on those of system (a). In the worst case, this operation requires checking whether there exists a solution for n systems of nm equations in n unknowns with complexity $O(n^4m)$. Since the operation must be performed for every $a \in A$, the overall complexity of checking for consistency is $O(kn^4m)$. The reason why we have at most n systems of nm equations in n unknowns is as follows. As explained above, in the worst case, $|E| = n$ and $|P| = nm$. Let $P = \{u_1, \dots, u_{nm}\}$ and $E = \{e_1, \dots, e_n\}$. Then system (a) is of the following type:

$$\beta_1(S_{u_1}, e_1) + \cdots + \beta_{nm}(S_{u_{nm}}, e_1) = 0$$
$$\cdots$$
$$\beta_1(S_{u_1}, e_n) + \cdots + \beta_{nm}(S_{u_{nm}}, e_n) = 0.$$

For each of the n equations of system (b) of the type

$$\beta_1(S_{u_1}, ae_i) + \cdots + \beta_{nm}(S_{u_{nm}}, ae_i) = 0,$$

we have to check whether the coefficients (S_{u_k}, ae_i) depend linearly on those of system (a), i.e., we have to check whether there exists a solution for the following system of nm equations in n unknowns:

$$\lambda_1(S_{u_1}, e_1) + \cdots + \lambda_n(S_{u_1}, e_n) = (S_{u_1}, ae_i)$$
$$\cdots$$
$$\lambda_1(S_{u_{nm}}, e_1) + \cdots + \lambda_n(S_{u_{nm}}, e_n) = (S_{u_{nm}}, ae_i).$$

- Filling the table is a task of lower complexity with respect to those considered above and does not influence the final result. In fact, in the worst case, the table is of size $|E||P| = n^2m$. Each multiplicity query for filling one entry of the table will be for a string of length at most $m + 2n + 1$. In fact, strings in P get as long as m when a counterexample is found, and one character may be added to them at most n times when closing the table. Strings in E are obtained by adding a one-character prefix to strings previously added to E, and this may be done at most n times. An extra character $a \in \Sigma$ is added at the time of the multiplicity query between a string in P and a string in E. This is done for all $a \in \Sigma$. The overall complexity of filling the table is $O(kn^2m(n + m))$.

Since the main loop is repeated at most n times, the overall complexity of the algorithm is $O(kn^5m^2)$. Together with the fact that when the main loop terminates, $M \equiv S$, this establishes our main result.

THEOREM 4.5. *Recognizable $\mathbb{Q}$-sets may be exactly identified in polynomial time from queries and counterexamples.*

This may be seen as a generalization of Angluin's result for finite-state automata [3]. It should be noted that Theorem 4.1 was the inspiration behind the algorithm but has not been used to obtain the above result. In particular, the completeness of P was not directly verified in the algorithm nor used in the proofs. However, when the algorithm terminates, the set P must be complete, as shown below.

5. Complete sets of strings. If $u \in P$, because tables are kept closed during the execution of the learning algorithm, for any $a \in A$,

$$S_{ua} \equiv_E \sum_{v \in P} \hat{\mu}(a)_{u,v} S_v. \tag{18}$$

In order to proceed with our discussion, we need an assumption that is quite obvious: if also $ua \in P$, then the linear dependencies for S_{ua} are chosen in the easiest way, i.e., with

$$\hat{\mu}(a)_{u,ua} = 1 \quad \text{and} \quad \hat{\mu}(a)_{u,v} = 0 \quad \text{for } v \neq ua. \tag{19}$$

Equation (19) is acceptable without loss of generality because it provides a hypothesized value $\hat{\mu}$ that satisfies (18). If $ua \notin P$, then any choice of $\hat{\mu}$ is acceptable as long as it satisfies (18). If this choice is made, the following holds.

LEMMA 5.1. *Let $u \in P$, and suppose that when $ua \in P$, $\hat{\mu}(a)_{u,v}$ is chosen as in equation* (19), *then for any $v \in P$, $\hat{\mu}(u)_{\epsilon,v} = 1$ if $v = u$ $\hat{\mu}(u)_{\epsilon,v} = 0$ otherwise.*

Proof (by induction on $|u|$).

Base. Since $\hat{\mu}$ is a morphism, $\hat{\mu}(\epsilon)$ must be the identity matrix, i.e., $\hat{\mu}(\epsilon)_{\epsilon,v} = 1$ when $v = \epsilon$ and is 0 elsewhere, which is what we need to prove.

Inductive step. Let $u = xa$, where $x \in P$ as $u \in P$ and P is prefix-closed.

$$\hat{\mu}(xa)_{\epsilon,v} = \sum_{w \in P} \hat{\mu}(x)_{\epsilon,w} \hat{\mu}(a)_{w,v}. \tag{20}$$

By the inductive hypothesis, $\hat{\mu}(x)_{\epsilon,w} = 0$ except when $w = x$, where it is equal to 1. Then the right-hand side of the previous equation reduces to

$$\hat{\mu}(x)_{\epsilon,x} \hat{\mu}(a)_{x,v} = 1 * \hat{\mu}(a)_{x,v}. \tag{21}$$

By (19) and because $xa = u \in P$, this is equal to 1 when $v = xa$ and is 0 elsewhere. □

LEMMA 5.2. *Under the same assumptions as in Lemma* 5.1, *for* $u \in P$, $\hat{\mu}(uw)_{\epsilon,v} = \hat{\mu}(w)_{u,v}$.

Proof.

$$\hat{\mu}(uw)_{\epsilon,v} = \sum_{z \in P} \hat{\mu}(u)_{\epsilon,z} \hat{\mu}(w)_{z,v}. \tag{22}$$

By the previous lemma, the right-hand side is equal to

$$\hat{\mu}(u)_{\epsilon,u} \hat{\mu}(w)_{u,v} = 1 * \hat{\mu}(w)_{u,v}. \quad □ \tag{23}$$

THEOREM 5.3. *Suppose that when* $ua \in P$, $\hat{\mu}(a)_{u,v}$ *is chosen as in equation* (19). *Then when* $M \equiv S$,

$$S_{ua} \equiv \sum_{v \in P} \hat{\mu}(a)_{u,v} S_v. \tag{24}$$

Consequently, when the guess M is correct, P must be complete.

Proof. We shall show that both sides of (24), when applied to any string $x \in A^*$, produce the same value.

$$\begin{aligned} (S_{ua}, x) &= (S, uax) = (M, uax) \\ &= \sum_{v \in P} \hat{\mu}(uax)_{\epsilon,v} (S_v, \epsilon) \\ &= \sum_{v \in P} \hat{\mu}(ax)_{u,v} (S_v, \epsilon) \quad \text{(by Lemma 5.2)}. \end{aligned} \tag{25}$$

$$\begin{aligned} \sum_{v \in P} \hat{\mu}(a)_{u,v} (S_v, x) &= \sum_{v \in P} \hat{\mu}(a)_{u,v} (S, vx) = \sum_{v \in P} \hat{\mu}(a)_{u,v} (M, vx) \\ &= \sum_{v \in P} \hat{\mu}(a)_{u,v} \sum_{w \in P} \hat{\mu}(vx)_{\epsilon,w} (S_w, \epsilon) \\ &= \sum_{v \in P} \hat{\mu}(a)_{u,v} \sum_{w \in P} \hat{\mu}(x)_{v,w} (S_w, \epsilon) \quad \text{(by Lemma 5.2)} \\ &= \sum_{w \in P} \sum_{v \in P} \hat{\mu}(a)_{u,v} \hat{\mu}(x)_{v,w} (S_w, \epsilon) = \sum_{w \in P} \hat{\mu}(ax)_{u,w} (S_w, \epsilon). \end{aligned}$$

This is equal to (25) after substituting v for w. □

Therefore, the completeness of P may be seen as a characterization of success when we learn that S, at least when $\hat{\mu}(ua)$ for $ua \in P$, is chosen as to verify (19). Moreover, it may be noted that if we start with a prefix-closed set P which is already complete, by virtue of Theorem 4.1, S may be exactly identified in polynomial time by means of multiplicity queries only. The algorithm is as follows:

1. $\mathcal{T} \leftarrow (P, \{\epsilon\}, T)$, where T is filled in with multiplicity queries.
2. Make the table $\mathcal{T}$ consistent, and fill in missing values with queries.
3. Output $M(\mathcal{T})$.

The table will certainly be closed since P is complete, and by Theorem 4.1, the final guess M must be correct. This result should be compared with [2], where a similar framework is described for finite-state automata: a regular language may be exactly identified in polynomial time using only membership queries, if we are given a complete set of representatives for Nerode's equivalence classes.

6. PAC-learnability and extensions. The learning method above also leads to some PAC-learnability results. If we do not require exact identification of the target $\mathbb{Q}$-set but are only interested in PAC-learnability, equivalence queries may be eliminated. Instead of asking an equivalence query, the algorithm will sample example strings and check whether the $\mathbb{Q}$-set learned at some stage is correct for these strings. The technique follows strictly from that used for DFAs in [3]. Consequently, $\mathbb{Q}$-automata are polynomially PAC-learnable when multiplicity queries are allowed. As a further consequence, nondeterministic automata may be PAC-predicted in polynomial time with multiplicity queries. It should be noted that negative results have been proved if only membership queries are available [4]. We do not have the proper PAC-learnability of NFAs with multiplicity queries because the representation that is learned is a $\mathbb{Q}$-automaton, not an NFA.

If a nondeterministic automaton is unambiguous, the corresponding $\mathbb{Q}$-set S will be such that for any string w, (S, w) is either 0 or 1. In this case, then, multiplicity queries reduce to ordinary membership queries. Now suppose that a regular language L is recognized by an unambiguous NFA M with a corresponding $\mathbb{Q}$-set S. This paper gives an algorithm for PAC-learning a representation of S in polynomial time with respect to the number of states of M if membership queries are allowed. In other words, regular languages are polynomially *predictable* using membership queries with respect to the representation of unambiguous nondeterministic automata. The importance of this lies in the fact that there are unambiguous NFAs such that the equivalent DFA has an exponentially larger number of states [16]. We then have a substantial improvement over previous results that established predictability with respect to a deterministic representation. It should be emphasized that the result only holds for *unambiguous* NFAs, and general NFAs are not predictable with membership queries, as shown in [4].

PAC-predictability is also established for probabilistic automata if multiplicity queries are allowed. As explained in §2, probabilistic automata may be represented by particular $\mathbb{Q}$-automata, and therefore they may be PAC-predicted with multiplicity queries. Again, we do not prove proper PAC-learnability because the learned representation is not a probabilistic automaton. In this case, multiplicity queries correspond to asking for the exact probability of accepting a given string. Previous results [17] required stronger types of queries. An interesting open problem is whether the present algorithm can be extended to the case where the oracle only provides an approximate probability of accepting some string. This would be more natural since one could think of estimating the probability by reading the string with the target probabilistic automaton several times.

Acknowledgments. We would like to thank the anonymous referees for the many remarks that helped us improve upon the initial version of this article.

REFERENCES

[1] D. ANGLUIN, *On the complexity of minimum inference of regular sets*, Inform. and Control, 39 (1978), pp. 337–350.

[2] ———, *A note on the number of queries needed to identify regular languages*, Inform. and Control, 51 (1981), pp. 76–87.

[3] ———, *Learning regular sets from queries and counterexamples*, Inform. and Comput., 75 (1987), pp. 87–106.

[4] D. ANGLUIN AND M. KHARITONOV, *When won't membership queries help?*, in Proc. 23rd Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1991, pp. 444–454.

[5] D. Angluin, *Negative results for equivalence queries*, Mach. Learning, 5 (1990), pp. 121–150.

[6] F. Bergadano and D. Gunetti, *Inductive Logic Programming: From Machine Learning to Software Engineering*, MIT Press, Cambridge, MA, 1996.

[7] F. Bergadano and S. Varricchio, *Learning behaviours of automata from multiplicity and equivalence queries*, in Proc. 1994 Italian Conference on Algorithms and Complexity, Lecture Notes in Comput. Sci., 778, Springer-Verlag, Berlin, New York, Heidelberg, 1994, pp. 54–62.

[8] J. Berstel and C. Reutenauer, *Rational Series and Their Languages*, Springer-Verlag, Berlin, 1988.

[9] S. Eilenberg, *Automata, Languages and Machines*, Vol. A, Academic Press, New York, 1974.

[10] M. E. Gold, *Complexity of automaton identification from given data*, Inform. and Control, 37 (1978), pp. 302–320.

[11] T. Harju and J. Karhumaki, *Decidability of the multiplicity equivalence problem of multitape finite automata*, Proc. 22nd Symposium on the Theory of Computing, Association for Computing Machinery, New York, 1990, pp. 477–481.

[12] B. K. Natarajan, *Machine Learning: A Theoretical Approach*, Morgan Kaufmann, San Mateo, CA, 1991.

[13] A. Paz, *Introduction to Probabilistic Automata*, Academic Press, New York, 1991.

[14] L. Pitt and M. K. Warmuth, *The minimum consistent DFA problem cannot be approximated within any polynomial*, J. Assoc. Comput. Mach., 40 (1993), pp. 95–142.

[15] A. Salomaa and M. Soittola, *Automata Theoretic Aspects of Formal Power Series*, Springer-Verlag, New York, 1978.

[16] R. E. Stearns and H. B. Hunt, *On the equivalence and containment problems for unambiguous regular expressions, regular grammars, and finite automata*, SIAM J. Comput., 14 (1985), pp. 598–611.

[17] W. Tzeng, *Learning probabilistic automata and markov chains via queries*, Mach. Learning, 8 (1992), pp. 151–166.

[18] S. Varricchio, *On the decidability of the equivalence problem for partially commutative rational power series*, Theoret. Comput. Sci., 99 (1992), pp. 291–299.