Sequential and Parallel Complexity of Learning DFA *

José L. Balcázar *Josep Díaz* *Ricard Gavaldà*
Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica Catalunya
Pau Gargallo 5, 08028 Barcelona, Spain

Osamu Watanabe
Department of Computer Science
Tokyo Institute of Technology
Meguro-ku, Tokyo 152, Japan

Abstract. It is known that the class of deterministic finite automata is polynomial time learnable by using membership and equivalence queries. We investigate the query complexity of learning deterministic finite automata, i.e., the number of membership and equivalence queries made during the process of learning. We prove lower bounds on the number of alternations between membership and equivalence queries, and also show that a trade-off exists, allowing us to reduce the number of equivalence queries at the price of increasing the number of membership queries. Finally, we study learning in a parallel model, the CRCW PRAM. We prove a lower bound on the parallel time needed for learning and design an algorithm that asymptotically achieves this bound.

1. Introduction

Query learning was introduced by Angluin [1] and is currently one of the most important models in computational learning theory. It differs from other models, such as inductive

*This research was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM). The first three authors were supported in part by the DAAD through Acciones Integradas 1992, 131-B, 313-AI-e-es/zk. The fourth author was supported in part by Takayanagi Foundation of Electronics and Science Technology. E-mail addresses: balqui@lsi.upc.es, diaz@siva.upc.es, gavalda@lsi.upc.es, watanabe@cs.titech.ac.jp

inference or certain PAC-learning algorithms, in that the learning process, the *learner*, obtains information about the concept to learn by making *queries* to some *teacher*, instead of passively receiving examples.

Variants of the formalization of learning via queries have been proposed in [14, 15]. We are interested here in the notion of "bounded learning" described there. In bounded learning, the learning algorithm is given a number as an input parameter. The goal of the algorithm is to output some hypothesis that only needs to be correct up to the length indicated by the input parameter. This learning notion is somewhat different from the original notion studied in Angluin's papers, but it allows us to avoid tedious and minor problems in the original notion. (See [15] for the justification of the bounded learning notion.)

It must be mentioned that all the concept classes used in this paper are finite and have a fixed length, so that the negative results also hold under Angluin's learning notion. Additionally, our learning algorithms for positive results also achieve exact learning.

The formalization of the concept of learning is particularly useful since a substantial gain of understanding comes from the possibility of relating it to various concepts from Computational Complexity. In particular, there are negative results for learning that rely on widely believed complexity-theoretic hypothesis, such as $R \neq NP$ or the existence of cryptographic one-way functions [10, 4]. Additionally, the idea of considering queries as a resource allows one to prove absolute negative results, whose proofs are independent of the learner's computational power: they are based instead on bounding the number of queries asked, and do not rely on any assumption. This contrasts with the negative results that depend on additional hypothesis. All our results here are absolute in this sense.

One of the successful fields in query learning is the problem of constructing a deterministic finite automaton (henceforth dfa) from information about the set it accepts. This is the problem we study in this paper. Pitt [11] surveys the status of this important problem in several learning models. For the case of query learning, Angluin proved an important positive result:

Proposition 1.1. [1] There exists a polynomial time algorithm that constructs a dfa using membership and equivalence queries.

Algorithms using these two kinds of queries will be called here *(Mem,Equ)-learners*. When only membership or only equivalence queries are allowed to be asked by the algorithm, we will call it a (Mem)-learner, respectively (Equ)-learner. Angluin showed that

neither membership queries alone nor equivalence queries alone are good enough to learn dfa in polynomial time.

Proposition 1.2.
(1) [2] No polynomial time (Mem)-learner exists for dfa.
(2) [3] No polynomial time (Equ)-learner exists for dfa.

Actually, it is easy to see that Angluin's dfa learning algorithm (witnessing Proposition 1.1) needs at most n equivalence queries for learning n state dfa. In this paper we investigate the *query complexity* of learning algorithms for dfa, i.e. the number of membership and equivalence queries used or needed to learn a fixed dfa.

First, we study the number of times that a learner must alternate between membership and equivalence queries. We impose limitations on this "alternations" resource, as a natural generalization of the learners that only use one of the two sorts of queries — which alternate 0 times—, and show that this restriction implies an increase of the total number of queries. For instance, as a consequence of this result, we show that in order to learn an n-state dfa in polynomial time, membership and equivalence queries must alternate at least $\Omega(n/\log^2 n)$ times.

Second, we study whether it is possible to reduce the number of one type of queries, maybe at the expense of the other. The learning formalism does not take into account how the queries are answered; but it is intuitively clear that, for many representation classes, answering a membership query can be substantially easier than answering an equivalence query. For instance, in the dfa case, evaluating a dfa on a word is one of the simplest problems in complexity theory, while deciding the equivalence of two dfa is complete for nondeterministic logspace. We prove that the number of such expensive queries can be reduced to some extent. More precisely, when no restriction is assumed on the way membership and equivalence queries are distributed on the computation of the learner, and only a bound on their number is set, a certain type of "trade-off" between them occurs. We show that it is possible to reduce the number of equivalence of queries, say, to $n/f(n)$ while increasing that of membership queries by a factor of $2^{f(n)}$. On the other hand, we also prove that in order to reduce equivalence queries to $n/f(n)$ one has to increase the number of membership queries by a factor of $2^{\Omega(f(n))}$. Thus, the above $2^{f(n)}$ increase is essential. For example, we can construct a polynomial time dfa learning algorithm that asks $n/c\log n$ equivalence queries, but it is impossible to reduce equivalence queries more than a factor of $O(\log n)$ without using a superpolynomial number of membership queries.

Finally, we consider whether parallel models can achieve substantially better running times in learning. We consider a well-studied model of parallel machine, the Concurrent-Read Concurrent-Write PRAM. Extending the lower bound obtained for the sequential model, we show that any CRCW PRAM using a polynomial number of processors requires $\Omega(n/\log n)$ time for learning DFA. Then we give an upper bound that matches this lower bound (up to multiplicative constants), i.e., a parallel version of Angluin's algorithm with $O(n/\log n)$ running time and polynomially many processors.

Results reported in sections 3 and 4 were reported in [5]. Results in section 5 will appear in [6].

2. Preliminaries

In this paper we follow standard definitions and notations in formal language theory and computational complexity theory; in particular, those for finite automata are used without definition. The reader will find them in standard textbooks such as [8].

Let Σ denote $\{0,1\}$, and throughout this paper, we use Σ as our alphabet. For any set A of strings, let $\overline{A}$ to denote the complement of A, i.e., $\Sigma^* - A$. For any sets A and B of strings, let $A \triangle B$ denote the set $(A - B) \cup (B - A)$. The length of a string x is denoted by $|x|$. The cardinality of a finite set A is written as $\|A\|$. Symbols $A^{\leq m}$ and $A^{=m}$ are used to denote the sets $\{x \in A : |x| \leq m\}$ and $\{x \in A : |x| = m\}$ respectively.

Notions and Notations for Query Learning

We briefly explain the notions and notations for discussing query learning formally. We basically follow the style established in [14, 15].

A learning problem is specified as a "representation class" [12]. A *representation class* is a triple (R, Φ, ρ), where $R \subseteq \Sigma^*$ is a *representation language*, $\Phi : R \to 2^{\Sigma^*}$ is a *semantic function* or *concept mapping*, and $\rho : R \to \mathbf{N}$ is a *size function*. For example, a representation class for dfa[1] is formally defined as follows: $DFA = (R_{\mathrm{dfa}}, \Phi_{\mathrm{dfa}}, \rho_{\mathrm{dfa}})$, where R_{dfa} is the set of dfa that are encoded in Σ^*, and for any $r \in R_{\mathrm{dfa}}$, $\Phi_{\mathrm{dfa}}(r)$ and $\rho_{\mathrm{dfa}}(r)$ are respectively the regular language accepted by the dfa (represented by) r and the number of states in the dfa (represented by) r. Following common convention, we write $\Phi_{\mathrm{dfa}}(r)$ as $L(r)$ and $\rho_{\mathrm{dfa}}(r)$ as $|r|$.

The encoding R_{dfa} is assumed to be honest, i.e. not much longer than necessary; in particular, we assume that the encoding of a dfa is polynomially long in the number of states.

[1] By "deterministic finite automaton" we mean a "complete" deterministic finite automaton over Σ^*. That is, we assume that the transition function is total.

Our computation model for learning is the "learning system". A *learning system* $\langle S, T \rangle$ is formed by a *learner* S and a *teacher* T. The learner has access to an input tape, an output tape, a communication tape, and a number of work tapes. The teacher has access to the same input and communication tapes, and furthermore a target tape.

The tapes except the communication tape and the target tape are used in a ordinary way. The communication tape is a read-write tape and used for the communication between S and T. That is, the queries from S and the answers from T are written on it. The target tape is a read-only tape, and its role is to let the teacher T know a *target concept*, a set to be learned. That is, a representation r of a target concept is written on the target tape; this situation intuitively means that T knows the concept that is represented by r. A teacher T who knows r (or, more precisely, T with r on the target tape) is written as "$T(r)$". Prior to the execution, the input ω and a target representation r are given respectively on the input tape and the target tape. Then the computation of $\langle S, T \rangle$ (which is written as $\langle S, T(r) \rangle(\omega)$) starts from S, executes S and T in turn, and finally halts at S. If S outputs y on its output tape and halts normally, then we say that $\langle S, T(r) \rangle(\omega)$ *outputs* y (and write $\langle S, T(r) \rangle(\omega) = y$).

In our framework, T is regarded as a function while S is regarded as some algorithm, or a Turing machine. That is, we omit considering T's computation and assume that T can *somehow* answer to queries.

For query types, we consider membership query (Mem) and equivalence query (Equ). For each membership query, the teacher T is supposed to answer "yes" or "no"; on the other hand, for each equivalence query, T is supposed to answer "yes" or provide a counterexample to the query. A learner is called, e.g., *(Mem,Equ)-learner* if it asks membership and equivalence queries, and a teacher is called, e.g., *(Equ)-teacher* if it answers only to equivalence queries. A tuple such as (Mem,Equ) is called a *query-answer* type[2].

Now we are ready to define our "learnability" notion. To simplify our discussion, we explain and define notions by using (Mem,Equ) for a typical query-answer type. However, these notions are defined similarly for other query-answer types.

In this paper, we consider only "bounded learning", which has been introduced in [14, 15] as one reasonable query learning notion. Intuitively, in the bounded learning, for a given parameter $m \geq 0$, the goal of a learner is to obtain a representation that

[2]The notation for query-answer types used in [14, 15] is more complicated in order to denote a finer query-answer type classification, including other query types. However, such classification is not necessary here; thus, we use this simpler notation.

denotes a target set up to length m. The parameter m is called a *length bound.* On the other hand, though we assume that teachers provide correct answers up to a given length bound, answers may not be correct if they are out of the length bound. By considering length bounds, we can avoid many tedious difficulties that come with the original and more general learning notion. Furthermore, bounded learning is well-motivated, and it is not just an artificial notion. Thus we use this learning notion throughout this paper. (Hence "bounded" is often omitted.) It should be noted, however, that every proof in this paper works even in the original query learning notion.

Let us define "bounded learning" more precisely. For any target representation r and, for a given equivalence query r', we say that $T(r)$ answers r' correctly up to length m if T gives a counterexample if it exists in $\Sigma^{\leq m}$ and answers "yes" otherwise. A teacher T is called a (*consistent*) *bounded (Mem,Equ)-teacher* for DFA if for given target representation r and length bound m, $T(r)$ answers each membership query correctly w.r.t. r, and $T(r)$ answers each equivalence query correctly up to length m. By considering a bounded teacher, we can avoid the case where a learner is given unnecessarily long counterexamples and the case where a learner abuses the teacher's power of searching through an infinite number of strings.

The value of m will be provided to the learning system as a part of the common input. Another part of the common input will be a value n, which is understood as a bound on the size of the output description to be written by the learner. This convention allows us to measure the time bound in terms of the input, as is customary in complexity theory.

Definition 2.1. [15] A (Mem,Equ)-learner S *learns* $C = (R, \Phi, \rho)$ (or C is *learned* by S) *in the bounded learning sense* if for every bounded (Mem,Equ)-teacher T for C, every $r \in R$, every $n \geq \rho(r)$, and every $m \geq 0$,

$$\langle S, T(r)\rangle(n, m) = r' \text{ such that } \Phi(r')^{\leq m} = \Phi(r)^{\leq m}.$$

Remark.

(1) Notice that the definition does not include the case where $n < \rho(r)$ is given as input for learning r. In other words, a learner can output anything in such a case. Thus, for specifying a learning algorithm, it is enough to consider the case that $n \geq \rho(r)$.

(2) In the later discussion, we assume that some additional parameter is given as an input. In such a case, the above and following definitions are extended naturally.

Now define the polynomial time learnability in the bounded learning sense. A learner is *polynomial time* if for some polynomial p and for all inputs $\langle n, m\rangle$, it halts within $p(n+m)$

steps. A representation class C is *polynomial time (Mem,Equ)-learnable in the bounded learning sense* if C is learnable by some polynomial time (Mem,Equ)-learner.

Finally, we define "query complexity". Intuitively, the "query complexity" is the number of queries asked by S in the worst case. More precisely, for any learner S for *DFA*, the query complexity $\#query_S$ is defined as follows: Let $\mathcal{T}$ be the family of bounded teachers for *DFA* of S's query-answer type. For any $T \in \mathcal{T}$, any $r \in R_{\rm dfa}$, and any $n, m \geq 0$, let $\#query_{\langle S,T(r)\rangle}(n,m)$ be the number of queries asked during the computation $\langle S,T(r)\rangle(n,m)$. Now for any $n, m \geq 0$,

$$\#query_S(n,m) \;=\; \max\{\; \#query_{\langle S,T(r)\rangle}(n,m) \;:\; T \in \mathcal{T}, r \in R_{\rm dfa} \;\}.$$

We are also interested in the alternation complexity of a (Mem,Equ)-learner S. Let $\mathcal{T}$ be the family of bounded (Mem,Equ)-teachers For any $T \in \mathcal{T}$, any $r \in R_{\rm dfa}$, and any $n, m \geq 0$, let $\#alt_{\langle S,T(r)\rangle}(n,m)$ be the number of times that S changes from membership to equivalence queries or vice-versa during the computation $\langle S,T(r)\rangle(n,m)$. Now for any $n, m \geq 0$,

$$\#alt_S(n,m) \;=\; \max\{\; \#alt_{\langle S,T(r)\rangle}(n,m) \;:\; T \in \mathcal{T}, r \in R_{\rm dfa} \;\}.$$

3. Query Alternations

Here we consider the case where the number of alternations between membership and equivalence queries is limited.

Theorem 3.1. Let $n_{a,k}$ and $m_{a,k}$ denote $(3k^2+2)(a+1)$ and $2k^2(a+1)$ respectively. For every constant $c > 1$ there is some constant $c' > 0$ such that for every (Mem,Equ)-learner S for *DFA*, every a, and every sufficiently large k.

- either S on input $\langle n_{a,k}, m_{a,k}\rangle$ asks some equivalence query with a dfa of size greater than $2^{c'k}$,
- or $\#alt_S(n_{a,k}, m_{a,k}) \;\geq\; a+1$,
- or $\#equ\text{-}query_S(n_{a,k}, m_{a,k}) \;\geq\; c^k$,
- or $\#mem\text{-}query_S(n_{a,k}, m_{a,k}) \;\geq\; 2^k - 1$.

Remark. Proposition 1.2 (2) is proved as a special case of this theorem.

For the proof, we use the technique established in [3]. Thus, we first recall some definitions and facts from [3]. For any $k > 0$, and any i, $1 \leq i \leq n$, define $L(i,k)$ to be the set of strings of length $2k$ whose ith bit is equal to the $(k+i)$th bit. Consider any set $L(i_1,k)L(i_2,k)\cdots L(i_k,k)$, where $1 \leq i_1,...,i_k \leq k$. The words in this set have length $m_{0,k} = 2k^2$. It is easy to show that the set is accepted by some dfa with $n_{0,k} = 3k^2 + 2$ states. Let R_k denote the set of dfa representations r such that $|r| = n_{0,k}$ and $L(r) = L(i_1,k)L(i_2,k)\cdots L(i_k,k)$ for some $1 \leq i_1,...,i_k \leq k$. From the above discussion, any set of the form $L(i_1,k)L(i_2,k)\cdots L(i_k,k)$ has a dfa representation in R_k; thus, $\|R_k\| = k^k$.

The following lemma, which states the lower bound of the number of equivalence queries for learning $r \in R_k$, plays a key role for proving our theorem.

Lemma 3.2. For any constant $c > 1$ there is a constant $c' > 0$ with the following property. Let S be any (Equ)-learner for *DFA* such that S on input $\langle n_{0,k}, m_{0,k}\rangle$ never asks a dfa (as an equivalence query) with more than $2^{c'k}$ states. Then $\#equ\text{-}query_S(n_{0,k}, m_{0,k}) \geq c^k$, for all sufficiently large k.

Proof. The proof is immediate from the argument in [3]. Here we review some important facts and state the proof outline.

Let S be defined as in the lemma, and let $c > 1$ be any constant. For any $k > 0$, define the following sets:

$$A_k = \{\, x_1x_1x_2x_2\cdots x_kx_k \;:\; \forall i, 1 \leq i \leq k\, [\, x_i \in \Sigma^k \,]\,\},$$
$$B_{c,k} = \{\, x_1y_1x_2y_2\cdots x_ky_k \;:\; \forall i, 1 \leq i \leq k\, [\, x_i, y_i \in \Sigma^k \wedge d(x_i,y_i) > (1-1/c)k \,]\,\}.$$

where $d(x,y)$ is the Hamming distance of x and y.

The following facts can be shown as in [3].

Fact 1.

(1) $A_k = \bigcap\{\, L(r) \;:\; r \in R_k \,\}$.

(2) For any $w \in B_{c,k}$, $\|\{\, r \in R_k \;:\; w \in L(r)\,\}\| \leq \left(\frac{k}{c}\right)^k$.

Fact 2. For every c there is some $c' > 0$ such that for any sufficiently large k, and for any dfa M with at most $2^{c'k}$ states, if M accepts all strings in A_k, then it accepts some string in $B_{c,k}$.

We define a teacher T_1 that answers an equivalence query $r \in R_{\text{dfa}}$ in the following way:

- So long as there are any "positive" counterexamples from A_k (i.e., strings in $A_k - L(r)$), T_1 returns one of them as a counterexample.
- Else if there are any "negative" counterexamples from $B_{c,k}$ (i.e., strings in $L(r) - B_{c,k}$), then T_1 returns one of them as a counterexample.
- Otherwise, T_1 returns some counterexample within the length bound, or returns "yes" if no counterexample exists.

Let k be any sufficiently large integer for which Fact 2 holds. We show that S needs to ask at least c^k equivalence queries to learn *some* $r_* \in R_k$ from T_1, where r_* will be determined through our discussion. Now consider the execution of $\langle S, T_1(r_*)\rangle(n_{0,k}, m_{0,k})$. This process is regarded as identifying r_* among the potential candidates. Clearly, at the beginning, every $r \in R_k$ is candidate, and for getting a correct answer for r_*, it is necessary to reduce the number of candidates to 1. We show that in order to achieve this goal, the execution needs at least c^k equivalence queries for some r_*. (In the following discussion, we assume that a candidate set is a subset of R_k. A real candidate set may contain other representations, but this only increases the number of queries.)

Let r_1 be S's first query in the execution. Suppose (the dfa represented by) r_1 does not accept some strings in A_k. Then T_1 returns one of them w as a positive counterexample. (I.e., w witnesses $L(r_*) - L(r_1) \neq \emptyset$.) But every $r \in R_k$ accepts w, so we cannot reduce the number of candidates by this counterexample. On the other hand, suppose that r_1 accepts every string in A_k. Then from Fact 2, it must accept some string u in $B_{c,k}$. Here we can assume that r_* is chosen so that it does not accept u. (Because the number of $r \in R_k$ that accepts u is at most $(k/c)^k << \|R_k\|$.) That is, $(L(r_1) - L(r_*)) \cap B_{c,k}$ contains at least one element, i.e., u. Then T_1 answers one of them w as a negative counterexample. (I.e., w witnesses $L(r_1) - L(r_*) \neq \emptyset$.) But by this counterexample, we can reduce the number of candidates by at most $(k/c)^k$. For, w is a negative counterexample to at most $(k/c)^k$ representations in R_k. By a similar argument, the second, third, ... counterexamples kill at most $(k/c)^k$ candidates each (if r_* is chosen appropriately). Thus, after q queries, at least $k^k - q(k/c)^k$ candidates are left. Hence, in order to have $k^k - q(k/c)^k \leq 1$, q must satisfy $q \geq c^k - (c/k)^k > c^k - 1$. That is, $q \geq c^k$. □

Proof of Theorem 3.1. Assume for contradiction that there is an integer a such that for some (Mem,Equ)-learner S and for infinitely many k, we have

- S on input $\langle n_{a,k}, m_{a,k}\rangle$ never asks a dfa with more than $2^{c'k}$ states,
- $\#alts_S(n_{a,k}, m_{a,k}) < a + 1$,

- $\#equ\text{-}query_S(n_{a,k}, m_{a,k}) < c^k$, and
- $\#mem\text{-}query_S(n_{a,k}, m_{a,k}) < 2^k - 1$.

Select such a to be the minimum with this property. We will contradict this minimality.

First observe that a is not 0. For $a = 0$, no alternation occurs. Thus, the learner is either a (Mem)-learner or an (Equ)-learner, and the lower bound follows from the results in [2] and [3]. More precisely, for (Mem)-learners, considering the class of dfa accepting a set of the form $\{w\}$ for some string w of length $k+1$, at least $2^k - 1$ queries are necessary. On the other hand, for (Equ)-learners, considering the class R_k, at least c^k queries are necessary. Notice that these dfa have at most $n_{0,k}$ states and accept only strings of length $m_{0,k}$. Thus, both lower bound results hold for any input $\langle n_{0.k}, m_{0.k}\rangle$ if k is sufficiently large.

Now we have that $a \geq 1$. Let $R_{a,k}$ be the set of all dfa representations with at most $n_{a,k}$ states accepting only strings of length at most $m_{a,k}$. We distinguish two cases depending on the type of the first query of S on input $\langle n_{a,k}, m_{a.k}\rangle$.

Case 1: membership queries are asked first.

Consider sets of representations for languages of the form $L = wL(r)$, where $r \in R_{a-1,k}$ and $|w| = k$. These representations have size at most $k+2+n_{a-1,k} \leq n_{a,k}$, and the length of the strings they accept is bounded by $k + m_{a-1,k} \leq m_{a,k}$. Hence, they are in $R_{a,k}$.

Simulate the initial membership query phase of S answering always "no". The number of queries is less than the total number of membership queries, so at the end of the phase we can select some w that has never appeared as prefix of a query. If S is still able to learn the representations for $wL(r)$ when $r \in R_{a-1,k}$, then a trivial modification of S learns $R_{a-1,k}$ with $a-1$ query alternations, which contradicts the minimality of a.

Case 2: equivalence queries are asked first.

We now consider representations for sets of the form $0L(r) \cup 1L(r')$, where $r \in R_{a-1,k}$ and r' is in R_k, the class used in Lemma 3.2. The representations for $0L(r) \cup 1L(r')$ have size at most $n_{a-1,k} + (3k^2+2) = n_{a,k}$, and the length of the strings they accept is bounded by $m_{a-1,k} + 1 \leq m_{a,k}$. Hence, they are in $R_{a,k}$ again.

For the first phase of equivalence queries of S, use a teacher that answers while possible with counterexamples for the $1L(r')$ part. By the bound on the number of equivalence queries, we know that after this phase there remain at least two representations in R_k that S cannot distinguish. Moreover, during the process, S has obtained only counterexamples beginning with 1. If it is able now to learn the part $0L(r)$, then a trivial modification

learns $R_{a-1,k}$ in $a-1$ alternations, contradicting again the minimality of a. □

The following negative result is easy to obtain from the theorem.

Theorem 3.3. There is no polynomial time (Mem,Equ)-learner for *DFA* that alternates $o(\frac{n}{\log^2 n})$ times between membership and equivalence queries.

Proof. Consider any infinite sequence $\{n_i\}_{i\geq 0}$ of natural numbers such that for each n_i there are a_i and k_i with the properties:

- $n_i = (3k_i^2+1)\cdot(a_i+1)$,
- $a_i = o(n_i/\log^2 n_i)$ (as a function of i).

Note that $k_i = \omega(\log n_i)$. We will show that no polynomial time (Mem,Equ)-learner for DFA can alternate less than a_i times to learn dfa with n_i states. The theorem follows if we can prove this for any sequence $\{n_i\}_{i\geq 0}$ with these properties.

Take any learner S that runs in polynomial time, and for each n_i in the sequence consider the behavior of S with input $\langle n_i, 2k_i(a_i+1)\rangle$. By Theorem 3.1, one of the following facts holds for some target dfa with n_i states:

1. either S asks an equivalence query of size at least $2^{c'k_i}$,
2. or S alternates more than a_i times,
3. or S asks at least c^{k_i} equivalence queries,
4. or S asks at least $2^{k_i}-1$ membership queries,

where $c>1$ and $c'>0$ are constants. If cases 1, 3, or 4 hold for infinitely many i, then the running time of S is $d^{k_i} = d^{\omega(\log n_i)}$, for infinitely many i and the constant $d = \min\{2^{c'}, c, 2\} > 1$. This contradicts the assumption that S runs in polynomial time. Hence, case 2 must hold for all but finitely many i and we are done. □

4. Trade-off Between the Number of Membership and Equivalence Queries

In this section, we consider the general case; that is, no restriction (except the number of queries) is assumed on the way of asking membership and equivalence queries. We show some trade-off relation between the number of membership and equivalence queries.

Let us consider the performance of Angluin's query learning algorithm for *DFA*. Suppose that the algorithm is to learn a n state dfa within a length bound m. Then it is easy to see that the algorithm asks at most n equivalence queries and a polynomial number of membership queries. Here we improve the equivalence query complexity while spending some more membership queries. More specifically, our improved algorithm takes $\langle n, m, h\rangle$ as input and learns a target dfa in the bounded learning sense, while asking n/h equivalence queries and $2^h \cdot p_1(n+m)$ membership queries, where p_1 is some fixed polynomial. Furthermore, the algorithm runs in polynomial time w.r.t. the number of queries.

Theorem 4.1. There is a (Mem,Equ)-learner S_0 for *DFA* with the following complexity: for every $n, m, h > 0$,
(a) $\#equ\text{-}query_{S_0}(n,m,h) \leq \frac{n}{h}$,
(b) $\#mem\text{-}query_{S_0}(n,m,h) \leq 2^h \cdot p_1(n+m)$, and
(c) S_0 on input $\langle n, m, h\rangle$ halts within time $p_2(\#query_{S_0}(n,m,h))$,
where p_1 and p_2 are polynomials depending on S_0.
Remark. The upper bound for the membership query complexity depends on the choice of our alphabet, i.e., $\Sigma = \{0,1\}$. More in general, we have $\#mem\text{-}query_{S_0}(n,m,h) \leq \|\Sigma\|^h \cdot p_1'(n+m+\|\Sigma\|)$.

Proof. We first recall some facts about Angluin's algorithm. In the algorithm, an *observation table* plays an important role for constructing hypotheses. An observation table is a tuple (S, E, T), where S and E are finite and prefix-closed sets, and T maps $(S \cup S\cdot\Sigma)\times E$ to $\{0,1\}$. At certain points, the algorithm builds a dfa $M = M(S,E,T)$ (i.e., a hypothesis) from the table, and presents M to the teacher as an equivalence query. If the answer is "yes", the algorithm halts. Otherwise, it uses a received counterexample to expand S, E, and T, in a way such that the next equivalence query must have at least one more state than the previous one. Furthermore, the algorithm has the following property: If the target set is accepted by a n state dfa, then when the constructed hypothesis has n states at some point, it must accept exactly the target language. From these properties, it is clear that Angluin's algorithm needs at most n equivalence queries.

We design S_0 so that it adds at least h states to M between each two consecutive equivalence queries. If we can still ensure that n bounds the number of states of M, then clearly S_0 needs at most n/h equivalence queries for obtaining the target dfa.

To do this, we define the notion of *observation table with lookahead h*. It is a tuple (S, E, T) as before, but table T maps $(S \cdot \Sigma^{\leq h}) \times E$ to $\{0,1\}$. Note that Angluin's tables are tables with lookahead 1. S_0 acts as Angluin's algorithm, but keeping an observation

table with lookahead h. This requires filling $\|\Sigma^{\leq h}\|$ entries in T with membership queries each time that S increases.

Clearly, this modification does not affect the correctness of the learner; that is, like Angluin's learning algorithm, S_0 learns DFA correctly. Furthermore, maintaining this additional information roughly increases the number of necessary membership queries by $\|\Sigma^{\leq h}\| \cdot p_1(n+m)$ for some polynomial p_1. Thus, the entire membership query complexity satisfies the theorem with some polynomial p_1. It is also easy to show that S_0 halts in time polynomial in the total number of queries.

Now it remains to show that at least h states are added after each equivalence query since this implies the desired upper bound on the number of equivalence queries. To show this property, it is enough to prove the following stronger version of Lemma 4 in [1]. □

Lemma 4.2. Assume that (S, E, T) is a closed and consistent observation table with lookahead h. Suppose that dfa $M = M(S, E, T)$ has k states. If M' is any dfa consistent with T that has less than $k + h$ states, then M' is isomorphic to M.

Proof. In the following, let $M = (Q, q_0, F, \delta)$ and $M' = (Q', q'_0, F', \delta')$. We assume without loss of generality that M' is minimum. That is, every state of M' can be reached from q'_0 and no two states in M' are equivalent.

We show an isomorphism between M and M'. Let us recall of give some definitions first.

- Recall that for a string $s \in S \cdot \Sigma^{\leq h}$, $row(s)$ denotes the finite function mapping each $e \in E$ to $T(s, e)$. Recall also that $Q = \{row(s) : s \in S\}$ is the set of states of M.
- For every $q' \in Q'$, define $Row(q')$ to be a finite function from E to $\{0, 1\}$ such that $Row(q')(e) = 1$ iff $\delta'(q', e) \in F'$.
- For every $s \in S$, define $f(s) = \delta'(q'_0, s)$. Note that $Row(f(s)) = row(s)$, from the assumption that M' is consistent with T. In fact, for every $u \in \Sigma^{\leq p}$, $Row(\delta'(f(s), u)) = row(s \cdot u)$.
- For every $q \in Q$, define $\phi(q) = \{f(s) : row(s) = q\}$.

In the following sequence of claims, we show that ϕ defines a bijection between Q and the set $\{\{q'\} : q' \in Q'\}$. Clearly, we can then transform ϕ into a bijection from Q to Q', and this turns out to be our desired isomorphism.

Claim 1. $\|Range(f)\| \geq k$.

Proof. From the above remark, it is easy to see that $f(s_1) = f(s_2)$ implies $row(s_1) = row(s_2)$. On the other hand, there are k different rows $row(s)$ in T (i.e., the states of M): hence, there must be at least k different $f(s)$. Thus, $\|Range(f)\| \geq k$. □ Claim 1

Intuitively, the next claim states that for any two states in M' there is already some string in E that proves them different. Here is where we make explicit use of the lookahead.

Claim 2. For any two different states q'_1 and q'_2 in Q', $Row(q'_1) \neq Row(q'_2)$.

Proof. By induction on the length of a string x witnessing that q'_1 and q'_2 are not equivalent.

If x is the empty string, then $Row(q'_1)$ and $Row(q'_2)$ are different in the entry corresponding to the empty string.

Consider the case where x is not empty. For the first symbol $a \in \Sigma$ of x, define $q'_3 = \delta'(q'_1, a)$ and $q'_4 = \delta'(q'_2, a)$. Then $q'_3 \neq q'_4$ (otherwise x is not a witness), and a string shorter than x witnesses that q'_3 and q'_4 are not equivalent. By induction hypothesis, $Row(q'_3)$ is different from $Row(q'_4)$.

On the other hand, because there are less than $k + h$ states in $Q' - Range(f)$ (since $\|Range(f)\| \geq k$ from Claim 1), q'_1 and q'_2 must be reachable from states in $Range(f)$ with a path of length less than h. More precisely, there exist u, v in $\Sigma^{<h}$ and s_1, s_2 in S such that $q'_1 = \delta'(f(s_1), u)$ and $q'_2 = \delta'(f(s_2), v)$. Then

$$row(s_1 \cdot ua) = Row(q'_3) \neq Row(q'_4) = row(s_2 \cdot va)$$

(the equalities are true because M' is consistent with T and ua and va are in $\Sigma^{\leq k}$). By the consistency of T, it must happen that

$$row(s_1 \cdot u) \neq row(s_2 \cdot v).$$

But $row(s_1 \cdot u) = Row(q'_1)$ and $row(s_2 \cdot v) = Row(q'_2)$, again because M' is consistent with T, and u and v are in $\Sigma^{\leq p}$. Hence $Row(q_1) \neq Row(q_2)$. □ Claim 2

Now using Claim 2, we prove that ϕ is a bijection from Q to $\{\{q'\} : q' \in Q'\}$ in the following way.

Claim 3.

(1) For every $q \in Q$, $\|\phi(q)\| \leq 1$,

(2) $Q' \subseteq Range(f)$, and

(3) ϕ is a bijection from Q to $\{\{q'\} : q' \in Q'\}$.

Proof. Part (1): Suppose that some $\phi(q)$ has two different states q'_1 and q'_2 in Q'. By Claim 2, $Row(q'_1) \neq Row(q'_2)$. Since both q'_1 and q'_2 are in $\phi(q)$, there are strings s_1, s_2 in S such that $q'_1 = f(s_1)$, $q'_2 = f(s_2)$, and $row(s_1) = row(s_2) = q$. However, we have $row(s_1)$ $(= Row(f(s_1))) = Row(q'_1) \neq Row(q'_2) = (Row(f(s_2)) =) row(s_2)$. A contradiction.

Part (2): Take any q' in Q'. By an argument as in Claim 2, q' must be reachable from some state $f(s_1)$ using a string $u \in \Sigma^{<h}$, that is, $Row(q') = row(s_1 \cdot u)$. Because T is closed, there is some $s_2 \in S$ such that $row(s_1 \cdot u) = row(s_2)$. Therefore, $Row(q') = row(s_2) = Row(f(s_2))$. By Claim 2 this means $q' = f(s_2)$, and thus $q' \in Range(f)$.

Part (3): From the above part (2) and our definitions, we have

$$Q' \subseteq Range(f) \subseteq \bigcup_{q \in Q} \phi(q) \subseteq Q'.$$

Note also that $\|Range(f)\| \geq k$ (but $\|Q\| = k$) and that $\|\phi(q)\| \leq 1$ for every $q \in Q$. Thus, it must happen that $\|\phi(q)\| = 1$ for every $q \in Q$ and that all $\phi(q)$ are different. Furthermore, every $q' \in Q'$ has some $q \in Q$ such that $q' \in \phi(q)$, which is in fact $\{q'\} = \phi(q)$. □ Claim 3

Finally we must show that ϕ is not only a bijection but also an isomorphism between M and M'. That is, it carries q_0 to q'_0, it preserves δ to δ', and it carries F to F'. But having proved that Q and Q' have the same cardinality, the rest is exactly as in Angluin's proof. □

From this theorem, it is straightforward to derive the following two upper bound results.

Corollary 4.3. Let $f(n)$ be any polynomial time computable function such that $f(n) < n$. There exist a (Mem,Equ)-learner S_f for DFA, and a polynomial q_f such that for every $n, m > 0$,

$$\#equ\text{-}query_{S_f}(n,m) \leq \frac{n}{f(n)} \text{ and } \#mem\text{-}query_{S_f}(n,m) \leq 2^{f(n)} \cdot q_f(n+m).$$

Corollary 4.4. For any $c > 0$, there exists a polynomial time (Mem,Equ)-learner S_c for DFA such that $\#equ\text{-}query_{S_c}(n,m) \leq \frac{n}{c \log n}$ for every $n, m > 0$.

Concerning these upper bound results, a natural question is whether they can be improved. For example, we may ask whether the number of equivalence queries can be reduced by more than a $O(\log n)$ factor. However, it is shown in the following that such reduction is not possible without increasing the number of membership queries more than polynomially. That is, there is a certain type of trade-off between the number of membership and equivalence queries.

For showing such trade-off phenomena, we first prove the following somewhat general lower bound.

Theorem 4.5. For any (Mem,Equ)-learner S for *DFA*, and for any $n, m > 0$, we have the following bounds:

$$\#\textit{equ-query}_S(n,m) \geq \left\lfloor \frac{n-2}{m} \right\rfloor \text{ or } \#\textit{mem-query}_S(n,m) \geq 2^m - \left\lfloor \frac{n-2}{m} \right\rfloor.$$

Remark. Proposition 1.2 (1) is a special case, i.e., $n = m + 2$, of this theorem.

Proof. We show a property stronger than the theorem: For every (Mem,Equ)-learner S that has query complexity better than the above, and *every* (Mem,Equ)-teacher T, S fails to learn some dfa from T.

To prove this, fix a (Mem,Equ)-learner S, a bounded (Mem,Equ)-teacher T for *DFA*, and any $n, m > 0$. Without loss of generality we may assume that $m + 2 \leq n \leq m2^m + 2$ (otherwise one of the inequalities in the theorem holds trivially). Let $l = \lfloor (n-2)/m \rfloor$, and assume that $\#\textit{equ-query}_S(n,m) < l$ and $\#\textit{mem-query}_S(n,m) < 2^m - l$. Define $R_{l,m,n}$ to be the set of dfa representations r such that $|r| \leq n \wedge L(r) \subseteq \Sigma^m \wedge \|L(r)\| \leq l$. Notice that every $L \subseteq \Sigma^m$ such that $\|L\| \leq l$ is accepted by some dfa with at most $lm + 2 \leq n$ states. Thus, any $L \subseteq \Sigma^m$ of size at most l has some dfa representation in $R_{l,m,n}$.

We prove that S fails to learn some dfa representation in $R_{l,m,n}$, using an adversary argument. Define two sets *Pos* and *Neg* to be initially empty. Simulate S with input $\langle n, m \rangle$, answering its queries as follows:

- When S makes a membership query x, answer "yes" if $x \in \textit{Pos}$; otherwise, answer "no" and add x to *Neg*.
- When S makes equivalence query r, one of the following three cases occurs.
  - $\textit{Pos} \neq L(r)^{\leq m}$: Return the counterexample x given by T for $L(r)$ when the target concept is *Pos*, and add x to *Neg* if it is not in *Pos*.
  - $\textit{Pos} = L(r)^{\leq m}$ and there is some x of length m not in $\textit{Pos} \cup \textit{Neg}$: Return x and add x to *Pos*.
  - Otherwise: Return "yes".

Let $\textit{Pos}_i$ be the value of *Pos* after the adversary has answered the ith query of S. We make the following claim, whose verification is straightforward and left to the reader.

Claim. For every i, the answers returned by the adversary for the first i queries are exactly those returned by the teacher T when the target concept is $\textit{Pos}_i$.

With this claim we can prove that the simulation of S with the adversary makes less than l equivalence queries. Assume otherwise that S makes l (or more) equivalence queries to the adversary. By the claim, it also makes l equivalence queries to the teacher T. Let

Pos_u be the value of Pos when (more precisely, just before) S makes its lth equivalence query. Note that strings are added to Pos only when S makes equivalence queries, so $\|Pos_u\| \leq l-1$. Furthermore, we always have $Pos_u \subseteq \Sigma^m$. Hence, Pos_u has some dfa representation in $R_{l,m,n}$ of size at most n. That is, S makes l or more equivalence queries to T on some target concept in $R_{l,m,n}$. A contradiction with the query bound we have assumed for S.

Similarly, we can prove that S makes less than $2^m - l$ membership queries to the adversary. Thus, the total number of queries to the adversary is at most $(l-1) + (2^m - l - 1) = 2^m - 2$.

Let POS and NEG be the values of Pos and Neg when S halts. Note that each query of S adds at most one string to either POS or NEG, that is, $\|POS \cup NEG\| \leq 2^m - 2$. Hence, there are two different strings $w_1, w_2 \in \Sigma^m$ that are not in $POS \cup NEG$. Note that $POS \cup \{w_1\}$ and $POS \cup \{w_2\}$ has at most l elements and that they are subsets of Σ^m; thus, some dfa $r_1, r_2 \in R_{l,m,n}$ recognize these two sets. Now, it follows from our discussion above that S receives the same answers during the executions of $\langle S, T(r_1)\rangle(n,m)$ and $\langle S, T(r_2)\rangle(n,m)$, namely, the answers given by the adversary. Therefore, S with teacher T outputs a wrong representation for either r_1 or r_2. □

As a corollary of this theorem, we have the following lower bound in contrast with Corollary 4.3.

Corollary 4.6. Let $f(n)$ be any function such that $f(n) < n$ and $f(n)$ becomes arbitrarily large as n increases, and let S be any (Mem,Equ)-learner for DFA. Then for some constants $c_1, c_2 > 0$, and for infinitely many $n, m > 0$, we have

$$\#equ\text{-}query_S(n,m) \geq \frac{n}{f(n)} \text{ or } \#mem\text{-}query_S(n,m) \geq 2^{c_1 f(n)} - \frac{c_2 n}{f(n)}$$

Proof. Define a nondecreasing sequence $m_1, m_2, \ldots$ so that $m_n \geq f(n)/2$ and $n/f(n) \leq \lfloor (n-2)/m_n \rfloor \leq 3n/f(n)$. Then the corollary follows from Theorem 4.5 for these pairs of n and m_n. (In this rough estimation, $c_1 = 1/2$ and $c_2 = 3$.) □

Thus, roughly speaking, the reduction of the equivalence query complexity by $1/f(n)$ factor always costs us about $2^{O(f(n))}$ membership queries. One interesting example is the following case, which shows the limitation of Corollary 4.4.

Corollary 4.7. Let $f(n)$ be any function such that $f(n) < n$ and $f(n)/\log n$ becomes arbitrarily large as n increases. Then there exists no polynomial time (Mem,Equ)-learner S for DFA with the query complexity $\#equ\text{-}query_S(n,m) \leq \frac{n}{f(n)}$.

5. Learning in parallel

The parallel computation model we consider is the Concurrent-Read Concurrent PRAM (CRCW PRAM). See for example [9] for a description. Since each PRAM memory cell contains an integer, we assume that a string over $\Sigma = \{0, 1\}$ is represented in a PRAM memory as a range of memory cells containing consecutive bits of the string, i.e., 0/1 integers. Hence, the string can be identified by its start and end addresses.

Individual PRAM processors can make different queries at each time step. To make query, the processor presents the teacher with the start and end memory addresses of the queried string as well as the addresses where the answer is to be received.

We first prove our lower bound on the parallel time for learning DFA using two main ideas. First, the lower bound on the number of Equivalence queries needed for sequential learning, i.e., Corollary 4.7. Second, a trick introduced by Bshouty and Cleve [7] to reduce the number of Equivalence queries when transforming a parallel into a sequential learner.

Theorem 5.1. There is no CRCW PRAM (Mem,Equ)-learner for *DFA* that runs in time $o\left(\frac{n}{\log n}\right)$ with a polynomial number of processors.

Proof. We prove the theorem by contradiction. Assume that there is a CRCW PRAM that learns dfa in $o\left(\frac{n}{\log n}\right)$ parallel time and using a $p(n)$ processors, with p a polynomial. We use this PRAM to build a sequential learner for dfa that makes $o\left(\frac{n}{\log n}\right)$ Equivalence queries and runs in polynomial time. This contradicts Corollary 4.7.

So take the hypothetical PRAM. We can assume without loss of generality that at any particular step, either all processors compute without queries, or all of them ask queries of the same type. (Any PRAM can be turned into one with this property without increasing the number of processors and a constant slowdown factor.)

The sequential learner S is as follows. It simulates sequentially steps 1, 2, ... of the PRAM. If, at step i, processors compute, S simpy updates the PRAM memory. If, at step i, procesors ask Membership queries, S asks all of these queries sequentially and returns to each processor its corresponding answer. If, at step i, processors ask Equivalence queries $r_1, r_2, \ldots, r_{p(n)}$, it uses the technique in [BC91] to answer all of them by making $p(n) - 1$ Membership queries and a single Equivalence query.

To do this, find a string in the symmetric difference of $\Phi(r_1)$ and $\Phi(r_2)$. This can be done easily since r_1 and r_2 are dfa. Ask this string as a Membership query. Depending on the answer, the string can be used as a counterexample for either r_1 or r_2. Let r_{i_1}.

$i \in \{1,2\}$, be the query that did not get a counterexample. Find another string in $\Phi(r_{i_1}) \triangle \Phi(r_3)$, ask it for membership, obtain a counterexample for one of the dfa, and call the other r_{i_2}, $i_2 \in \{1,2,3\}$. And so on. After $p(n) - 1$ rounds, we are left with only one query $r_{i_{p(n)-1}}$, $i_{p(n)-1} \in \{1,\ldots,p(n)\}$, whose counterexample is obtained with an Equivalence query.

Simulating a parallel step of the PRAM clearly takes polynomial time, and there are $o(n/\log n)$ parallel steps. Hence, S runs in polynomial time. Also, S makes at most one Equivalence query per simulated step. This gives us the desired contradiction. □

Now we present a CRCW PRAM algorithm whose running time matches this lower bound. Thus, with respect to time, this algorithm is optimal.

To describe the algorithm, we borrow many concepts and notations from the proof of Theorem 4.1. In the following, let M_* be a given target dfa, and let L_* denote the lanuage accepted by M_*. Without loss of generality, we may assume that M_* is the minimum dfa.

Recall that, for an observation table (S, E, T) and a string $s \in S$, $row(s)$ is the row vector defined by $(T[s,e])_{e\in E}$, and for a string $e \in E$, $col(e)$ is the column vector defined by $(T[s,e])_{s\in S}$. Similarly, for any string w, $row_*(w)$ is the row vector defined by $(L_*[w,e])_{e\in E}$. We say that *column e separates rows s_1 and s_2* if $T[s_1,e] \neq T[s_2,e]$. A string $s \in S$ or $row(s)$ is called *essential* if there is no s' lexicographically smaller than s such that $row(s) = row(s')$. Similarly, we say that $e \in E$ or $col(e)$ is *essential* if it is the lexicographically smallest column that separates some two rows in the table. Rows or columns that are not essential are called *unessential*.

Figure 1 presents the learning algorithm OPT that achieves the optimal upper bound. Essentially, it uses the queries to maintain a closed and consistent observation table. Intuitively, the observation table has sufficient lookahead to ensure that only $O(n/\log n)$ Equivalence queries are made, as in Theorem 4.1. But this is not enough because between each two Equivalence queries there can be as much as $\Omega(n)$ iterations needed to make the table closed and consistent. This algorithm uses parallelism to solve in constant time chunks of about $\log n$ each iterations, thus reducing the number of iterations to $O(n/\log n)$.

We first detail the meaning and implementation of some of the lines in the algorithm. Using standard tricks of CRCW PRAM programming, it is possible to show that each of these lines requires only $O(1)$ time and a number of processors polynomial in n, $2^h = n$, and the length of the longest counterexample received so far. We omit the detailed descriptions in this version.

1. build initial table (S, E, T);
2. **while** true **do**
3. expand table (S, E, T);
4. remove useless rows and columns from (S, E, T);
5. **if** the table closed and consistent **then**
6. build $M(S, E, T)$;
7. ask $M(S, E, T)$ as Equivalence query;
8. **if** answer = 'YES' **then**
9. **output** $M(S, E, T)$ and **halt**
10. **else** (i.e., answer = $\langle \mathrm{NO}, w \rangle$ for some w)
11. add w to (S, E, T) **fi fi**
12. **od**.

Figure 1: Learning Algorithm *OPT*

Line 1 "build initial table": This means to set $S = \{\lambda\}$ and $E = \{\lambda\}$, and then to fill in the entry $T[\lambda, \lambda]$ of T by asking one Membership query "$\lambda \in L_*$?".

Line 3 "expand table (S, E, T)": This means to expand table (S, E, T) and make an *auxiliary table* (S, E, T_{aux}) by asking Membership queries. Table T_{aux} is a two-dimensional table that is used to check whether (S, E, T) is closed and consistent at Line 5. Similar to T, T_{aux} keeps $L_*[s \cdot a, e]$ for every $s \in S$, $a \in \Sigma$, and $e \in E$.

The detail of "expand table (S, E, T)" is the following steps.

for all pairs (s, u) with $s \in S$ and $u \in \Sigma^{\leq h}$ **do in parallel** add $s \cdot u$ to S;
for all pairs (e, v) with $e \in E$ and $v \in \Sigma^{\leq h}$ **do in parallel** add $v \cdot e$ to E;
for all pairs (s, e) with $s \in S$ and $e \in E$ **do in parallel**
fill the entry $T[s, e]$ of T by asking Membership query "$s \cdot e \in L_*$?";
for all triples (s, a, e) with $s \in S$, $a \in \Sigma$, and $e \in E$ **do in parallel**
fill the entry $T_{\mathrm{aux}}[s \cdot a, e]$ of T_{aux} by asking Membership query "$s \cdot a \cdot e \in L_*$?";

Line 4 "remove useless rows and columns": This means removing from S all elements such that: (i) they are unessential, and (ii) they are only prefixes of unessential elements in S. Remove from T all rows indexed by elements removed from S. All in all at most

each representative for each row is left, if we consider only the elements of S that are not prefixes of other elements in S. Useless columns are removed similarly. That is, "remove useless columns" means removing from E all elements such that: (i) they are unessential, and (ii) they are only suffixes of useless elements in E. Remove from T all columns indexed by elements removed from E.

Line 5 "if the table closed and consistent": This means to test whether (S, E, T) is closed and consistent with lookahead 1. The test can be done just following the definition by using auxiliary table (S, E, T_{aux}).

Line 10 "add w to (S, E, T)": This means that for each prefix u of w, add u to S as new rows of T.

Correctness of OPT.

Consider the execution of the above algorithm when target dfa M_* has n_* states. We prove that the algorithm terminates in $O(n_*/\log n_*)$ iterations, and then we are done because each iteration takes constant time by the comments above. Roughly speaking, we show that the size (the number of states) of minimum dfa that is consistent with the observation table increases at least by $h/2$ after each iteration. On the other hand, since M_* is the minimum dfa accepting L_* and the observationn table is consitent with L_*, n_* is an upper bound of the size of minimum dfa consistent with the table. Thus, the number of iterations must be bounded by $2n_*/h$, whichi is $O(n_*/\log n_*)$.

For the following lemmas, we introduce one more notion. First, let us assume, in the following discussion, that observation tables are consistent with the target L_*. For any $k \geq 1$ and any table (S, E, T), the *k-expansion* of (S, E, T) is a table (S', E', T') such that $S' = S \cup S \cdot \Sigma^{\leq k}$, $E' = E \cup \Sigma \cdot S^{\leq k}$, and T' is consistent with L_*. We say that the k-expansion of (S, E, T) to (S', E', T') is *flat* if no essential row is introduced in (S', E', T'), or more precisely, both (S, E, T) and (S', E', T') have the same number of essential rows. Here we should note that if the k-expansion of (S, E, T) to (S', E', T') is *flat* then (S, E, T) is closed and self-consistent.

Fact 3. For any $k \geq 1$ and any (S, E, T), if its k-expansion is flat, then (S, E, T) is closed and self-consistent.

Now the fact that the number of iterations is bounded by $2n_*/h$ follows from next two lemmas, whose proofs are not presented in this version (they will be produced real soon now).

Lemma 5.2. The total number of non-flat expansions in the sequential execution of *OPT* is at most n_*.

Lemma 5.3. The total number of flat expansions in the sequential execution of *OPT* is at most n_*.

Since each iteration of *OPT* simulates h iterations of its sequential execution, it follows that there are at most $2n_*/h$ iterations. Each iteration uses constant time using a number of processors polynomial in n, $2^h = n$, and the length of the longest counterexample received. Hence, we have the following result.

Theorem 5.4. There is a CRCW PRAM (Mem,Equ)-learner for DFA that uses $O(n/\log n)$ time and a polynomial number of processors.

References

[1] Angluin, D., Learning regular sets from queries and counterexamples. *Information and Computation*, Vol. 75, 1987, pp.87-106.

[2] Angluin, D., Queries and concept learning, *Machine Learning*. Vol. 2. 1988, pp.319–342.

[3] Angluin, D., Negative results for equivalence queries, *Machine Learning*, Vol. 5, 1990. pp.121–150.

[4] Angluin, D., and Kharitonov, M., When won't membership queries help?, in *Proc. 23rd ACM Symp. on Theory of Computing*, ACM, 1991. pp.444–454.

[5] Balcázar, J.L., Díaz, J., Gavaldà, R., and Watanabe, O.. A note on the query complexity of learning DFA, in *Proc. Third Workshop on Algorithmic Learning Theory*. Japanese Society for Artificial Intelligence, 1992, pp.53–62.

[6] Balcázar, J.L., Díaz, J., Gavaldà, R., and Watanabe, O., An optimal parallel algorithm for learning DFA, in preparation.

[7] Bshouty, N., and Cleve, R., On the exact learning of formulas in parallel, in *Proc. 33rd IEEE Symp. on Foundations of Computer Science*. IEEE. 1992. pp.513–522.

[8] Hopcroft, J, and Ullman, J., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979.

[9] Karp R.M. and V. Ramachandran, "Parallel algorithms for shared-memory machines", in *Handbook of Theoretical Computer Science*, vol. A, MIT/Elsevier, 1990, 869–941.

[10] Kearns, M., and Valiant, L., Cryptographic limitations on learning boolean formulae and finite automata, in *Proc. 21st ACM Symp. on Theory of Computing*, ACM, 1989, pp.433–444.

[11] Pitt, L., Inductive inference, DFAs, and computational complexity, in *Proc. International Workshop on Analogical and Inductive Inference AII'89*, Lecture Notes in Artificial Intelligence 397, Springer-Verlag, 1989, pp.18–42.

[12] Pitt, L., and Warmuth, M., Reductions among prediction problems: on the difficulty of prediction automata, in *Proc. 3rd Structure in Complexity Theory Conference*, IEEE, 1988, pp.60–69.

[13] Valiant, L., A theory of the learnable, *Communications of the ACM*, Vol. 27, 1984, pp.1134–1142.

[14] Watanabe, O., A formal study of learning via queries. in *Proc. 17th International Colloquium on Automata, Languages and Programming*. Lecture Notes in Computer Science 443, Springer-Verlag, 1990, pp.137–152.

[15] Watanabe, O., A framework for polynomial time query learnability, Technical Report 92TR-0003, Dept. of Computer Science, Tokyo Institute of Technology, 1992. To appear in *Mathematical Computation Theory*.