The Learnability of Simple Deterministic Finite-Memory Automata

Sakamoto, Hiroshi
Department of Informatics Kyushu University

TR-CS-127

DOI Technical Report

The Learnability of Simple Deterministic Finite-Memory Automata

Hiroshi Sakamoto

September 19, 1996

Department of Informatics
Kyushu University 33
Fukuoka 812-81, Japan
E-mail:hiroshi@i.kyushu-u.ac.jp Phone:092-641-1101 Ext.8424

The Learnability of Simple Deterministic Finite-Memory Automata

Hiroshi Sakamoto
Department of Informatics
Kyushu University 33, Fukuoka 812-81, Japan
email: hiroshi@i.kyushu-u.ac.jp

Abstract

The present paper establishes the learnability of simple deterministic finite-memory automata via membership and equivalence queries. Simple deterministic finite-memory automata are a subclass of deterministic finite-memory automata introduced by Kaminski and Francez [9] as a model generalizing finite-state automata to infinite alphabets.

For arriving at a meaningful learning model we first prove the equivalence problem for simple deterministic finite-memory automata to be decidable by reducing it to the equivalence problem for finite-state automata. In particular, we present a decision algorithm taking as input any two simple deterministic finite-memory automata $\mathcal{A}$ and $\mathcal{B}$ which computes a number k from $\mathcal{A}$ and $\mathcal{B}$ as well as two finite state automata $M_{\mathcal{A}}$, $M_{\mathcal{B}}$ over a finite alphabet Σ of cardinality k such that $\mathcal{A}$ and $\mathcal{B}$ are equivalent iff $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$ are equivalent over Σ.

Next, we provide the announced learning algorithm and prove its running time to be polynomially bounded in the length of a longest counter example returned, in k, the number described above, and in n the number of states of a minimum deterministic finite state automata being consistent with the target language over a finite alphabet of cardinality k.

1 Introduction

Active learning goes at least back to Shapiro [12], [13], [14] who developed an Algorithmic Debugging System that uses a variety of types of queries to the user to pinpoint errors in Prolog programs. Subsequently, Sammut and Banerji [11] proposed a learning system using membership queries. Theoretical investigation of learning via queries started with Angluin's [1] pioneering paper. In particular, Angluin [1] showed the class of regular languages represented by deterministic finite automata to be learnable by using polynomially many membership and equivalence queries. For facilating further discussion, we introduce some additional notation.

Let Σ be a finite alphabet, and let Σ^* be the set of all finite strings over Σ. Any $L \subseteq \Sigma^*$ is called a language. The general problem studied in active learning is to identify a target language L with respect to some hypothesis space $L_1, L_2, \ldots$, of languages over Σ. A membership query takes as input a string $s \in \Sigma^*$ and returns *yes* if $s \in L$ and *no* otherwise. An equivalence query takes as input a language $\hat{L}$ from the hypothesis space and returns *yes* if $\hat{L} = L$. Otherwise, *no* is returned as well as a counterexample from the symmetric difference of $\hat{L}$ and L. Clearly, in order to arrive at a meaningful model, one has to require that the languages in the hypothesis space do possess finite descriptions which are provided as input to an equivalence query. Therefore, we assume the hypotheses to be represented by its indices. The questions are assumed to be truthfully answered by a teacher. The learning task is successfully finished if the learning algorithm returns an index i such that $L = L_i$. Moreover, the relevant equivalence problems should be decidable, since otherwise the teacher would have to much power (cf. [5]). Finally, the overall number of queries ask should be polynomially bounded in some suitable parameters, e.g., the minimal index i with $L = L_i$, and the length of the longest counterexample returned.

During the last decade, various researchers continued along this line. Sakakibara [10] generalized Angluin's result to the class of bottom up tree automata. Ishizaka [8] showed that the class of simple deterministic context-free grammars is polynomially learnable from membership and equivalence queries. More recently, Burago [3] provided an algorithm for learning the class of structurally reversible grammars in polynomial time using membership and equivalence queries, and Bergadano and Varricchio [4] showed that recognizable series are identifiable using polynomially many equivalence queries and shortest counterexamples. For more information concerning recent results within this learning model the reader is referred to Gavalda's [6] excellent survey.

We aim to extend this line research into a direction previously not considered, i.e., we study the identification of languages defined over an infinite but countable alphabet $\hat{\Sigma} = \{a_i | i \in \mathbb{N}\}$. Recently, Kaminski and Francez [9] introduced a model of computation dealing with languages over $\hat{\Sigma}$, the so-called finite-memory automata. Finite-memory automata are a natural extension of finite state automata in that they accept precisely the regular languages when restricted to finite alphabets. For the sake of presentation, we continue with a description of their model. A finite-memory automaton is represented by 6-tuple $\mathcal{A} = \langle S, \mathfrak{u}^{\mathcal{A}}, q_0, \rho, \mu, F\rangle$, where S is a finite set of states, $q_0 \in S$ is the initial state, $\mathfrak{u}^{\mathcal{A}} \in (\hat{\Sigma} \cup \{\#\})^k$ is the initial assignment, $\rho \subseteq S \times \{i \in \mathbb{N} | 1 \leq i \leq k\}$ is the reassignment, and $\mu \subseteq S \times \{i \in \mathbb{N} | 1 \leq i \leq k\} \times S$ the transition relation. Finally, $F \subseteq S$ is the set of accepting states, and $\#$ is a special symbol not belonging to $\hat{\Sigma}$.

A finite-memory automaton $\mathcal{A}$ is equipped with an auxiliary tape consisting of k

windows each of which can contain precisely one symbol from $\hat{\Sigma} \cup \{\#\}$. The content of this tape is referred to as to an assignment, thus $u^{\mathcal{A}}$ is called initial assignment of $\mathcal{A}$. A pair of a state and an assignment is called configuration. For $\mathcal{A}$, the unique configuration $(q_0, u^{\mathcal{A}})$ is called the initial configuration of $\mathcal{A}$ and a configuration with the first component in F is called a final configuration of $\mathcal{A}$. Let $(p, \mathfrak{u})$ be an actual configuration of $\mathcal{A}$. Then, for an input symbol $a \in \hat{\Sigma}$, $\mathcal{A}$ may change its actual configuration as follows. First, it tests whether or not a is contained in some window of $\mathfrak{u}$. In case it is, say in the ith window of $\mathfrak{u}$, then, if $(p, i, q) \in \mu$, $\mathcal{A}$ may enter state q and the assignment $\mathfrak{u}$ remains unchanged, i.e., $(q, \mathfrak{u})$ is the new actual configuration. In the case a is not contained in u, first $j = \rho(p)$ is computed and a is placed into the jth window of $\mathfrak{u}$ resulting into the new assignment $\hat{\mathfrak{u}}$. Furthermore, if $(p, j, q) \in \mu$, then $\mathcal{A}$ may enter state q, and $(q, \hat{\mathfrak{u}})$ is $\mathcal{A}$'s new configuration.

A computation of $\mathcal{A}$ is a sequence of configurations of $\mathcal{A}$. A computation ending in a final configuration of $\mathcal{A}$ is called an accepting run of $\mathcal{A}$. We use $L(\mathcal{A})$ to denote the set of all strings over $\hat{\Sigma}$ that are accepted by $\mathcal{A}$ and refer to $L(\mathcal{A})$ as to the language accepted by $\mathcal{A}$. Similarly to finite-state automata deterministic finite-memory automata can be defined. A finite-memory automaton $\mathcal{A}$ is called *deterministic* if ρ is a mapping: $S \mapsto \{i \in \mathbb{N} | 1 \leq i \leq k\}$ and for each $p \in S$ and $1 \leq i \leq k$, there exists exactly one $q \in S$ such that $(p, i, q) \in \mu$.

Furthermore, a finite-memory automata $\mathcal{A}$ is called *simple* if $\mathfrak{u}^{\mathcal{A}} \in \{\#\}^k$. Our domain of interest is the subclass of simple deterministic finite-memory automata, denoted by dfma#.

In the following, we consider the problem of learning simple deterministic finite-memory automata via membership and equivalence queries. After a bit of reflection, it is easy to see that the set of all dfma# is recursively enumerable. Thus, we fix any recursive enumeration $(\mathcal{A}_i)_{i \in \mathbb{N}}$ as hypothesis space. Now, assuming a teacher choosing any $\mathcal{A}$ as target, the learner can ask membership and equivalence queries to obtain information about the object to be learned.

Next, we discuss how reasonable it is to assume the teacher to answer correctly membership and equivalence queries. For membership queries, it is easy to see whether or not any string from $\hat{\Sigma}$ is accepted by the dfma# $\mathcal{A}$ chosen by the teacher. Thus, it remains to consider the feasibility of answering equivalence queries. This problem has not been studied by Kaminski and Francez [9] and remained, as far as we know, open. Thus, we first study the decidability of the equivalence problem for dfma#'s. Roughly speaking, the first result of this paper is as follows: Given any two dfma#'s $\mathcal{A}$ and $\mathcal{B}$ such that the length of their initial assignments are $k_{\mathcal{A}}$ and $k_{\mathcal{B}}$, respectively, we compute two dfa's $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$ consistent with $\mathcal{A}$ and $\mathcal{B}$ over a finite alphabet of cardinality $k \geq k_{\mathcal{A}} + k_{\mathcal{B}}$, respectively. Then, we prove that $\mathcal{A}$ and $\mathcal{B}$ are equivalent over $\hat{\Sigma}$ if and only if $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$ are equivalent over Σ. Furthermore, the equivalence problem for deterministic finite-state automata is decidable in polynomial time, the equivalence problem for dfma#'s is decidable, too. Thus, we conclude that it is reasonable to allow polynomially many equivalence queries for learning dfma#'s.

Moreover, we provide an algorithm, denoted by **DFMA**, that identifies every dfam# using polynomially many membership and equivalence queries. Thereby, the polynomial takes as input the length of a longest counterexample returned, the number k described above as well as n, the number of states of a minimum deterministic finite-state automata being consistent with the target language $L(\mathcal{A})$ over the finite alphabet Σ of cardinality k. Note that the **DFMA** is partially based on Angluin's [1] observation table technique.

This paper is organized as follows. Section 2 provides all definitions and notations. The equivalence problem for dfma#'s is studied in Section 3. Section 4 deals with the learning algorithm mentioned above. The first subsection formalizes the learning problem and provides the algorithm **DFMA**. In Subsection 4.2. we prove the correctness of the **DFMA**, and the final subsection deals with its running time. Finally, in Section 5 we discuss open problems.

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers. An *alphabet* is a set of symbols. By $\hat{\Sigma}\{a_i | i \in \mathbb{N}\}$ we denote an infinite but countable alphabet. Let $\hat{\Sigma}^*$ be the free monoid over $\hat{\Sigma}$ (cf. [7]). The elements of $\hat{\Sigma}^*$ are called strings. Let $w \in \hat{\Sigma}^*$ be a string.; we use $|w|$ to denote the *length* of w and by $[w]$ the range of w is denoted. By *range*(w), we denote the set of symbols contained in w. The string ε of length 0 is referred to as to the empty string. Furthermore, for all $i = 1, \dots, |w|$, we use $w[i]$ for denoting the ith symbol in w. Additionally, the prefix of length i of w, $0 \leq i \leq |w|$, is denoted by $w(i)$. Let $n \in \mathbb{N}$; then we set $\hat{\Sigma}^n = \{w \in \hat{\Sigma}^*,\ |w| = n\}$. Moreover, we set $\hat{\Sigma}^+ = \hat{\Sigma}^* \setminus \{\varepsilon\}$.

Let $\Sigma \subset \hat{\Sigma}$ be a finite alphabet. By $||\Sigma||$, we denote the cardinality of Σ. A *language* is a subset of $\hat{\Sigma}^*$ (possibly, a subset of Σ^*). A *class* of languages is a collection of languages containing at least one nonempty language.

Let # be a special symbol not belonging to $\hat{\Sigma}$. An *assignment* is a string $x_1 x_2 \cdots x_n \in (\hat{\Sigma} \cup \{\#\})^*$ such that if $x_i = x_j$ and $i \neq j$, then $x_i = \#$ for $1 \leq i, j \leq n$. That is, an assignment is a string over $\hat{\Sigma} \cup \{\#\}$ such that each symbol in $\hat{\Sigma}$ appears at most one time.

Definition 1 ***(Kaminski and Francez [9]).*** *A finite-memory automaton is 6-tuple $\mathcal{A} = \langle S, q_0, \mathrm{u}, \rho, \mu, F \rangle$, where S is a finite set of states, $q_0 \in S$ is an initial state, $\mathrm{u} = x_1 x_2 \cdots x_k \in (\Sigma \cup \{\#\})^k$ is an initial assignment, ρ is a mapping: $S \mapsto \{1, 2, \dots, k\}$ called a reassignment, $\mu \subseteq S \times \{1, 2, \dots, k\} \times S$ is called a transition relation and $F \subseteq S$ is a set of final states.*

A pair (q, u) of a state $q \in S$ and an assignment $\mathrm{u} \in (\hat{\Sigma} \cup \{\#\})^k$ is called a *configuration.* An unique configuration (q_0, u) is called an *initial configuration*, and all the configurations with the first component in F are called *final configurations.* We define a relation $\vdash$ over configurations as follows: Let $\mathrm{u} = x_1 x_2 \cdots x_k$ and $\mathrm{u}' = y_1 y_2 \cdots y_k$ be assignments, and let $p, q \in S$. Then, $(p, \mathrm{u}) \vdash (q, \mathrm{u}') \Leftrightarrow^{\text{def}}$ there exists $a \in \hat{\Sigma}$ such that

1. if $a = x_i$ for some $1 \leq i \leq k$, then $\mathrm{u} = u'$ and $(p, i, q) \in \mu$, or

2. if $a \notin [u]$, then $\rho(p) = j \in \{1, 2, \cdots, k\}$, $y_j = a$, for each $1 \leq j' \neq j \leq k$, $y_{j'} = x_{j'}$, and $(p, j, q) \in \mu$.

Intuitively, let p and u be an actual state and an actual assignment of $calA$, respectively. Then, if a is equal to the i-th symbol of u and $(p, i, q) \in \mu$, then $\mathcal{A}$ may change its state to q, and otherwise $\mathrm{u}[\rho(p)]$ is replaced by a and if $(p, \rho(p), q) \in \mu$, then $\mathcal{A}$ may change its state to q. When necessary, we write $(p, \mathrm{u}) \vdash^a (q, \mathrm{u}')$ by specifying the symbol

a. The reflexive, transitive closure of $\vdash$ is denoted by $\vdash^*$. We say that $\mathcal{A}$ *accepts* $w = w_1 w_2 \cdots w_n \in \hat{\Sigma}^*$ if there exists a sequence of configurations $c_0, c_1, \ldots, c_n$ such that c_0 is the initial configuration, c_n is a final configuration and $c_{i-1} \vdash^{w_i} c_i$ for each $1 \le i \le n$. The set of all strings accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$. By $(\mathcal{A}(x), \mathrm{u}_x)$, we denote a configuration in $S \times (\hat{\Sigma} \cup \{\#\})^k$ of $\mathcal{A}$ such that $(q_0, \mathrm{u}) \vdash^x (\mathcal{A}(x), \mathrm{u}_x)$ for $x \in \hat{\Sigma}^*$.

Definition 2 ***(Kaminski and Francez [9]).*** *A finite-memory automaton $\mathcal{A} = \langle S, q_0, \mathrm{u}, \rho, \mu, F\rangle$ is called deterministic if ρ is everywhere defined, and for each $p \in S$ and each $1 \le i \le |u|$, there exists exactly one $q \in S$ such that $(p, i, q) \in \mu$.*

Definition 3 *A finite-memory automaton $\mathcal{A} = \langle S, q_0, \mathrm{u}, \rho, \mu, F\rangle$ is called simple if $\mathrm{u} \in \{\#\}^*$.*

A deterministic finite-memory automaton is written by a dfma. In particular, a simple dfma is written by a dfma#. By *DFMA*, we denote the class of languages accepted by dfma's over $\hat{\Sigma}$. Our domain of interest is a subclass of *DFMA*, denoted by *DFMA*$^\#$, the class of languages accepted by dfma#'s.

3 Deciding equivalence of dfma#'s

In this section, we discuss the decidability of equivalence for dfma#'s. Given two dfma#'s and a countable alphabet, the equivalence of the dfma#'s over the alphabet is defined as follows.

Definition 4 *Let $\mathcal{A}$ and $\mathcal{B}$ be any dfma#'s, and let $\hat{\Sigma} = \{a_0, a_1, \ldots, a_n, \ldots\}$. We say that $\mathcal{A}$ and $\mathcal{B}$ are equivalent over $\hat{\Sigma}$ if, for all $w \in \hat{\Sigma}^*$, $w \in L(\mathcal{A})$ iff $w \in L(\mathcal{B})$.*

We provide an algorithm **Dec** deciding whether or not $\mathcal{A}$ and $\mathcal{B}$ are equivalent over $\hat{\Sigma}$. The behavior of **Dec** is summarized as follows; Let $\mathcal{A} = \langle S^{\mathcal{A}}, \mathrm{u}^{\mathcal{A}}, q_0^{\mathcal{A}}, \rho^{\mathcal{A}}, \mu^{\mathcal{A}}, F^{\mathcal{A}}\rangle$ and $\mathcal{B} = \langle S^{\mathcal{B}}, \mathrm{u}^{\mathcal{B}}, q_0^{\mathcal{B}}, \rho^{\mathcal{B}}, \mu^{\mathcal{B}}, F^{\mathcal{B}}\rangle$. Let $k = |u^{\mathcal{A}}| + |u^{\mathcal{B}}|$. Then **Dec** takes $\mathcal{A}$, $\mathcal{B}$ and $\Sigma = \{a_0, a_1, \ldots, a_k\}$ as input. First, it computes a set of configurations of $\mathcal{A}$ and $\mathcal{B}$ defined on Σ as follows; For $\mathcal{A}$, let $C(\mathcal{A}) = S^{\mathcal{A}} \times (\Sigma \cup \{\#\})^{|u^{\mathcal{A}}|}$. Initially, let $C(\mathcal{A}, 0) = \{(q_0, \mathrm{u}^{\mathcal{A}})\}$. For $i \in \mathbb{N}$, we proceed inductively. Define $C(\mathcal{A}, i) = C(\mathcal{A}, i-1) \cup C(\mathcal{A})_{i-1}$, where $C(\mathcal{A})_{i-1} = \{c \in C(\mathcal{A}) | \exists a \in \Sigma, \exists c' \in C(\mathcal{A}, i-1) \text{ s.t } c' \vdash^a c\}$. If $C(\mathcal{A}, i) = C(\mathcal{A}, i-1)$ for some $i \ge 0$, then let $C(\mathcal{A}) := C(\mathcal{A}, i)$. For $\mathcal{B}$, a set of possible all configurations, $C(\mathcal{B})$, is analogously computed. From $C(\mathcal{A})$, a dfa $M_{\mathcal{A}} = \langle Q^{\mathcal{A}}, \Sigma, p_0^{\mathcal{A}}, \delta^{\mathcal{A}}, f^{\mathcal{A}}\rangle$ is defined as follows; $Q^{\mathcal{A}} := C(\mathcal{A})$, $p_0^{\mathcal{A}} := (q_0^{\mathcal{A}}, \mathrm{u}^{\mathcal{A}})$, $\delta^{\mathcal{A}} \subseteq C(\mathcal{A}) \times \Sigma \times C(\mathcal{A})$, where $(c, a, c') \in \delta^{\mathcal{A}}$ iff $c \vdash^a c'$. Furthermore, $f^{\mathcal{A}} \subseteq C(\mathcal{A})$, where $(p, \mathrm{u}) \in f^{\mathcal{A}}$ iff $p \in F^{\mathcal{A}}$. $M_{\mathcal{B}}$ is defined analogously. Now, using any standard algorithm for testing equivalence of two dfa's as a subroutine, **Dec** decides whether or not $L(M_{\mathcal{A}}) = L(M_{\mathcal{B}})$, and returns 'yes' and 'no', respectively. For the formal description of **Dec**, we refer to the reader to Figure 1. Next, we deal with the correctness of algorithm **Dec**.

Theorem 1 *Let $\mathcal{A}$ and $\mathcal{B}$ be any two dfma#'s, and let Σ be defined as above. On input $\mathcal{A}$, $\mathcal{B}$ and Σ, the algorithm **Dec** eventually terminates.*

```
Input: Two dfma#'s $\mathcal{A}$ and $\mathcal{B}$, and an alphabet $\Sigma = \{a_0, a_1, \dots, a_k\}$.
begin
  Let $C(\mathcal{A}, 0) = \{(q_0, \mathrm{u}^{\mathcal{A}})\}$.
  compute $C(\mathcal{A}, 1)$ and let $i = 1$.
  while($C(\mathcal{A}, i) \neq C(\mathcal{A}, i-1)$)do
    compute $C(\mathcal{A}, i+1)$ and $i := i + 1$.
  end while
  let $C(A) := C(\mathcal{A}, i)$.
  let $M_{\mathcal{A}} = \langle Q^{\mathcal{A}}, p_0^{\mathcal{A}}, \delta^{\mathcal{A}}, f^{\mathcal{A}} \rangle$ such that
    $Q^{\mathcal{A}} = C(\mathcal{A}, i)$, $p_0^{\mathcal{A}} = (q_0^{\mathcal{A}}, \mathrm{u}^{\mathcal{A}})$,
    $\delta^{\mathcal{A}} \subseteq C(\mathcal{A}) \times \Sigma \times C(\mathcal{A})$, where $(c, a, c') \in \delta^{\mathcal{A}}$ iff $c \vdash^a c'$, and
    $f^{\mathcal{A}} \subseteq C(\mathcal{A})$, where $(p, \mathrm{u}) \in f^{\mathcal{A}}$ iff $p \in F^{\mathcal{A}}$.
  compute $M_{\mathcal{B}}$.
  if($L(M_{\mathcal{A}}) = L(M_{\mathcal{B}})$)then
    output 'yes' and halt.
  if($L(M_{\mathcal{A}}) \neq L(M_{\mathcal{B}})$)then
    output 'no' and halt.
end
```

Figure 1: Algorithm **Dec**

Proof. Since Σ, $S^{\mathcal{A}}$ and $S^{\mathcal{B}}$ are finite, $C(\mathcal{A})$ and $C(\mathcal{B})$ are finite, too. Thus, $C(\mathcal{A}, i) = C(\mathcal{A}, i-1)$ as well as $C(\mathcal{B}, i) = C(\mathcal{B}, i-1)$ must eventually happen. By definition, $C(\mathcal{A}, i) = C(\mathcal{A}, i-1)$ implies $C(\mathcal{A}, i) = C(\mathcal{A}, i+m)$ for all $m \geq 0$. Thus, assuming $(q_0, \mathrm{u}^{\mathcal{A}}) \vdash^w (p, \mathrm{u})$ for some $w \in \Sigma^*$ directly results in $(p, \mathrm{u}) \in C(\mathcal{A}, |w|)$, a contradiction. Hence, for all $(p, \mathrm{u}) \in C(\mathcal{A})$, $(q_0, \mathrm{u}^{\mathcal{A}}) \vdash^* (p, \mathrm{u})$ iff $(p, \mathrm{u}) \in C(\mathcal{A})$. Consequently, $C(\mathcal{A})$ and $C(\mathcal{B})$ are computable. Finally, it is easy to see that $M_{\mathcal{A}}$ and $M_{\mathcal{B}}$ are computable, too. Hence, algorithm **Dec** terminates. $\square$

Next, we show the correctness.

Definition 5 *Let $\mathcal{A} = \langle S, \mathrm{u}, q_0, \rho, \mu, F \rangle$ be a dfma# and $\Sigma = \{a_0, a_1, \dots, a_k\}$ such that $k \geq |u|$. For $w \in \hat{\Sigma}^+$, we define sets $w_{\Sigma}^{\mathcal{A}} \subseteq \Sigma^{|w|}$ recursively as follows:*

$$a_{\Sigma}^{\mathcal{A}} = \Sigma \quad \textit{for each } a \in \hat{\Sigma}$$

$$(wa)_{\Sigma}^{\mathcal{A}} = \bigcup_{w' \in (w)_{\Sigma}^{\mathcal{A}}} \{w'a'\} \begin{cases} a' = u_{w'}[i] & \textit{if } a = u_w[i] \\ a' \in \Sigma \backslash [u_{w'}] & \textit{otherwise.} \end{cases}$$

Let us verify that $w_{\Sigma}^{\mathcal{A}}$ is well-defined. Every $a_{\Sigma}^{\mathcal{A}}$ is defined for each $a \in \hat{\Sigma}$. Let $w_{\Sigma}^{\mathcal{A}}$ be defined for each $w \in \Sigma^n$. If $a = u_w[i]$, then $a \in [w]$. Then, let $w = \alpha a \beta$ for some $\alpha \in \hat{\Sigma}^*$ and $\beta \in (\hat{\Sigma} \backslash \{a\})^*$. Since $w_{\Sigma}^{\mathcal{A}}$ is defined, $(\alpha a)_{\Sigma}^{\mathcal{A}}$ is also defined. Then, $\mathrm{u}_{\alpha a}[i] = a$. Thus, $\mathrm{u}_{\alpha' a'}[i] = a' \neq \#$ is defined. Since $a \notin [\beta]$ and $\mathrm{u}_w[i] = a$, if $w' = \alpha' a' \beta' \in w_{\Sigma}^{\mathcal{A}}$ ($|\beta'| = |\beta|$), then $a' \notin [\beta']$. It follows that $\mathrm{u}_{w'}[i] = a' \neq \#$. Thus $(wa)_{\Sigma}^{\mathcal{A}}$ is defined. On the other hand, let $a \notin [u_w]$. Since $||\Sigma|| = k+1 > |u|$, for each $w \in \hat{\Sigma}^*$, clearly, $\Sigma \backslash [u_w] \neq \emptyset$. Thus, $(wa)^{\mathcal{A}}_{\Sigma}$ is defined.

Lemma 1 *Let $w \in \hat{\Sigma}^+$ and $w' \in w^{\mathcal{A}}_{\Sigma}$. Then, for each $1 \leq i \leq |w|$ and $1 \leq j \leq |u|$, $w[i] = u_{w(i-1)}[j]$ iff $w'[i] = u_{w'(i-1)}[j]$, where $\mathfrak{u}_{w(0)} = u$.*

Proof. It is proved by induction on length of w. Since an initial assignment $\mathfrak{u}$ contains no symbol in $\hat{\Sigma}$, by the definition, it is clear that, for each $a \in \hat{\Sigma}$ and $a' \in a^{\mathcal{A}}_{\Sigma} = \Sigma$, $a \in [u]$ iff $a' \in [u]$. Assume the induction hypothesis on $\hat{\Sigma}^n$ $(n \geq 1)$. Let $wa \in \hat{\Sigma}^{n+1}$ and $w'a' \in (wa)^{\mathcal{A}}_{\Sigma}$. If $a \notin [u_w]$, then, by the definition, $a' \in \Sigma' \backslash [u_{w'}]$. It follows that $a' \notin [u_{w'}]$. Let $a = u_w[j]$ for some $1 \leq j \leq |u|$. By the induction hypothesis, $\mathfrak{u}_w[j] \neq \#$ iff $\mathfrak{u}_{w'}[j] \neq \#$. Thus, by the definition, $a' = u_{w'}[j] \neq \#$. □

Lemma 2 *For each $w \in \hat{\Sigma}^+$ and $w' \in w^{\mathcal{A}}_{\Sigma}$, $\mathcal{A}(w) = \mathcal{A}(w')$.*

Proof. It is proved by induction on length of w. Clearly, $\mathcal{A}(a) = \mathcal{A}(a')$ for each $a \in \hat{\Sigma}$ and $a' \in x^{\mathcal{A}}_{\Sigma} = \Sigma$. Assume the induction hypothesis on $\hat{\Sigma}^n$ $(n \geq 1)$. Let $\mathcal{A}(w) = \mathcal{A}(w') = p$ for $w \in \hat{\Sigma}^n$ and $w' \in w^{\mathcal{A}}_{\Sigma}$. Let $wa \in \hat{\Sigma}^{n+1}$ and $w'a' \in (wa)^{\Sigma}_{\mathcal{A}}$. Let $a \notin [u_w]$ and $\rho(p) = i$ for some $1 \leq i \leq |u|$. Then, $(wa)^{\mathcal{A}}_{\Sigma} = \cup_{w' \in w^{\mathcal{A}}_{\Sigma}, a' \in \Sigma \backslash [u_{w'}]} x'a'$. Thus, by Lemma 1, $(p, \mathfrak{u}_w) \vdash^{a}$ $(q, \mathfrak{u}_{wa})$ on $(p, i, q) \in \mu$ implies $(p, \mathfrak{u}_{w'}) \vdash^{a'} (q, \mathfrak{u}_{w'a'})$ on $(p, i, q) \in \mu$. Thus, $\mathcal{A}(wa) = \mathcal{A}(w'a') = q$. Let $a = u_w[j]$ for some $1 \leq j \leq |u|$. Then, $(wa)^{\mathcal{A}}_{\Sigma} = \cup_{w' \in w^{\mathcal{A}}_{\Sigma}} w' u_{w'}[j]$. Thus, by Lemma 1, $(p, \mathfrak{u}_w) \vdash^{a} (q, \mathfrak{u}_{wa})$ on $(p, j, q) \in \mu$ implies $(p, \mathfrak{u}_{w'}) \vdash^{u_{w'}[j]} (q, \mathfrak{u}_{w'a'})$ on $(p, j, q) \in \mu$. Thus, $\mathcal{A}(wa) = \mathcal{A}(w' u_{w'}[j]) = \mathcal{A}(w'a') = q$. Hence, $\mathcal{A}(w) = \mathcal{A}(w')$ for each $w \in \hat{\Sigma}^{n+1}$ and $w' \in w^{\mathcal{A}}_{\Sigma}$. □

Lemma 3 *Let $\mathcal{A}$ and $\mathcal{B}$ be dfma#'s. Let $\Sigma = \{a_0, a_1, \cdots, a_k\}$ such that $k \geq |u^{\mathcal{A}}| + |u^{\mathcal{B}}|$. Then, for each $w \in \hat{\Sigma}^+$, $w^{\mathcal{A}}_{\Sigma} \cap w^{\mathcal{B}}_{\Sigma} \neq \emptyset$*

Proof. It is proved by induction on length of w. Clearly, for each $w \in \hat{\Sigma}$, $w^{\mathcal{A}}_{\Sigma} = w^{\mathcal{B}}_{\Sigma} = \Sigma$. Assume the induction hypothesis on $\hat{\Sigma}^n$ $(n \geq 1)$. Let $wa \in \hat{\Sigma}^{n+1}$.
Case 1. Let $a \in [u^{\mathcal{A}}_w] \cap [u^{\mathcal{B}}_w]$. Then, there exists $w' \in w^{\mathcal{A}}_{\Sigma} \cap w^{\mathcal{B}}_{\Sigma}$ such that $w' u^{\mathcal{A}}_{w'}[j_1] \in (wa)^{\mathcal{A}}_{\Sigma}$ for $a = u^{\mathcal{A}}_w[j_1]$ and $w' u^{\mathcal{B}}_{w'}[j_2] \in (wa)^{\mathcal{B}}_{\Sigma}$ for $a = u^{\mathcal{B}}_w[j_2]$. Since $a \in [u^{\mathcal{A}}_w]$, we can assume that $w = \alpha a \beta$ for some $\alpha \in \hat{\Sigma}^*$ and $\beta \in (\hat{\Sigma} \backslash \{a\})^*$. By the induction hypothesis, there exists $\alpha' a' \beta' \in (\alpha a \beta)^{\mathcal{A}}_{\Sigma} \cap (\alpha a \beta)^{\mathcal{B}}_{\Sigma}$. Since $a \notin [\beta]$ and $a \in [u^{\mathcal{A}}_w] \cap [u^{\mathcal{B}}_w]$, $\mathfrak{u}^{\mathcal{A}}_{\alpha a}[j_1] = u^{\mathcal{A}}_w[j_1] = u^{\mathcal{B}}_{\alpha a}[j_2] = u^{\mathcal{B}}_w[j_2] = a$. Thus, by Lemma 1, $\mathfrak{u}^{\mathcal{A}}_{\alpha' a'}[j_1] = u^{\mathcal{A}}_{w'}[j_1] = u^{\mathcal{B}}_{\alpha' a'}[j_2] = u^{\mathcal{B}}_{w'}[j_2] = a'$. Thus, $(wa)^{\mathcal{A}}_{\Sigma} \ni w' u^{\mathcal{A}}_{w'}[j_1] = w'a' = w' u^{\mathcal{B}}_{w'}[j_2] \in (wa)^{\mathcal{B}}_{\Sigma}$.
Case 2. Let $a \notin [u^{\mathcal{A}}_w] \cup [u^{\mathcal{B}}_w]$. By the condition $||\Sigma|| = k+1 > |u^{\mathcal{A}}| + |u^{\mathcal{B}}|$, for each $w' \in w^{\mathcal{A}}_{\Sigma} \cap w^{\mathcal{B}}_{\Sigma}$, there exists $a' \in \Sigma$ such that $a' \notin [u^{\mathcal{A}}_{w'}] \cup [u^{\mathcal{B}}_{w'}]$. Thus, $(wa)^{\mathcal{A}}_{\Sigma} \ni w'a' \in (wa)^{\mathcal{B}}_{\Sigma}$.
Case 3. Let $a \in [u^{\mathcal{A}}_w] \backslash [u^{\mathcal{B}}_w]$. Since $a \in [w]$, let $w = \alpha a \beta$ for some $\alpha \in \hat{\Sigma}^*$ and $\beta \in (\hat{\Sigma} \backslash \{a\})^*$. By the induction hypothesis, let $\alpha' a' \in (\alpha a)^{\mathcal{A}}_{\Sigma}$. By Lemma 1, there exists $1 \leq j_1 \leq |u^{\mathcal{A}}|$ such that $\mathfrak{u}^{\mathcal{A}}_{\alpha a}[j_1] = a$ iff $\mathfrak{u}^{\mathcal{A}}_{\alpha' a'}[j_1] = a'$. Let $w' = \alpha' a' \beta' \in w^{\mathcal{A}}_{\Sigma}$. Since $a \notin [\beta]$, by Lemma 1, $a' \notin [\beta']$. Thus, $w'a' \in (wa)^{\mathcal{A}}_{\Sigma}$. Since $a \in [u^{\mathcal{B}}_{\alpha a}]$, we can assume that $a = u^{\mathcal{B}}_{\alpha a}[j_2]$ and $a' = u^{\mathcal{B}}_{\alpha' a'}[j_2]$. Since $a \notin [u^{\mathcal{B}}_w]$, by Lemma 1, $\mathfrak{u}^{\mathcal{B}}_{\alpha a}[j_2] = a \neq u^{\mathcal{B}}_w[j_2]$ implies $\mathfrak{u}^{\mathcal{B}}_{\alpha' a'}[j_2] = a' \neq u^{\mathcal{B}}_{w'}[j_2]$. Since $a' \notin [\beta']$, we have that $a' \notin [u^{\mathcal{B}}_{w'}]$. Thus, $w'a' \in (wa)^{\mathcal{B}}_{\Sigma}$. Consequently, $w'a' \in (wa)^{\mathcal{A}}_{\Sigma} \cap (wa)^{\mathcal{B}}_{\Sigma}$. Finally, we have that, for each $w \in \hat{\Sigma}^{n+1}$, $w^{\mathcal{A}}_{\Sigma} \cap w^{\mathcal{B}}_{\Sigma} \neq \emptyset$. The proof is completed. □

Theorem 2 *Let $\mathcal{A}$ and $\mathcal{B}$ be dfma#'s. Let $\Sigma = \{a_0, a_1, \cdots, a_k\}$ such that $k \geq |u^{\mathcal{A}}| + |u^{\mathcal{B}}|$. Then, $\Sigma^* \cap L(\mathcal{A}) = \Sigma^* \cap L(\mathcal{B})$ if and only if $L(\mathcal{A}) = L(\mathcal{B})$.*

Proof. The if-part is always clear, then we prove the only-if-part. Let $w \in L(\mathcal{A})$. By Lemma 2, $w_{\Sigma}^{\mathcal{A}} \subseteq L(\mathcal{A})$. Since $\Sigma^* \cap L(\mathcal{A}) = \Sigma^* \cap L(\mathcal{B})$, $w_{\Sigma}^{\mathcal{A}} \subseteq L(\mathcal{B})$. By Lemma 3, there exists $w' \in w_{\Sigma}^{\mathcal{A}} \cap w_{\Sigma}^{\mathcal{B}}$. By Lemma 2, $\mathcal{B}(w') = \mathcal{B}(w)$. Thus, $w \in L(\mathcal{B})$. The converse direction is analogous. □

4 Learnability of dfma#'s

We discuss the learnability of dfma#'s using membership and equivalence queries. This section contains three parts. In the first part, we provide an algorithm using membership and equivalence queries, and explain its behavior for a target language in $DFMA^{\#}$. In the last two-part, we analysis its correctness and running time. Finally, the main theorem of this paper is proved.

First, we define membership and equivalence queries for a target dfma# $\mathcal{A}^*$. Given $\hat{\Sigma}$, the followings are allowed;

Membership query: an input is $w \in \hat{\Sigma}^*$

a response is $\begin{cases} \text{yes} & \text{if } w \in L(\mathcal{A}^*) \\ \text{no} & \text{otherwise} \end{cases}$

Equivalence query: an input is a dfma#$\mathcal{A}$

a response is $\begin{cases} \text{yes} & \text{if } L(\mathcal{A}) = L(\mathcal{A}^*) \\ \text{a string in } L(\mathcal{A}) \oplus L(\mathcal{A}^*) & \text{otherwise} \end{cases}$

We note the result on Section 3. Then, the above queries are not stronger than that on regular languages. Our goal is to show that every $L \in DFMA^{\#}$ is learnable on hypotheses space dfam#'s using membership and equivalence queries in polynomial-time in the parameters r, n and m; r is the length of a shortest initial assignment of dfma# $\mathcal{A}$ such that $L = L(\mathcal{A})$, n is the number of states of a minimum dfa M such that $L(M) = \Sigma^* \cap L$, where $||\Sigma|| = r$, and m is the length of a longest counterexample returned so far.

4.1 Algorithm DFMA

The algorithm $DFMA$ shown in Figure 2 contains three procedures denoted by **Table**(,), **Cons**() and **Mini**(,). Angluin introduced the notion of *observation tables* and showed that every dfa is learnable in polynomial-time using membership and equivalence queries [1]. **Table** is just the Angluin's algorithm, that is, given a finite alphabet Σ and a set E of (counter)examples, **Table**(Σ, E) is a minimum dfa M such that $L(M) = \Sigma^* \cap L$ for a target L.

Given a minimum dfa M such that $L(M) = \Sigma^* \cap L$, the procedure $Cons$ shown in Figure 3 computes a dfma# $\mathcal{A}$ with an initial assignment of length $n = ||\Sigma||$ as follows: Let $M = \langle S, \Sigma, q_0, \delta, F \rangle$ and $\Sigma = \{a_1, a_2, \cdots, a_n\}$. Initially, a one-to-one mapping $\cup_{1 \leq i \leq n}\{a_i \mapsto i\}$ is fixed and set $\mathcal{A} = \langle S, \mathfrak{u}, q_0, \rho, \mu, F \rangle$ such that $\mathfrak{u} = \#^n$ and for each $1 \leq i \leq n$, $(p, i, q) \in \mu$ iff $\delta(p, a_i) = q$. Furthermore, $\rho \subseteq S \times \{1, 2, \ldots, n\}$ is defined as follows: Initially, let $\rho_0 = \emptyset$. For $i \leq n$, let $\rho_i = \rho_{i-1} \cup \rho'_i$, where $\rho'_i \subseteq S \times \{1, 2, \ldots, n\}$

such that $(p,i) \in \rho'_i$ iff $(p,i) \notin \rho_{i-1}$ and $\delta(q_0,w) = p$ for some $w \in \Sigma\backslash\{a_i\}^*$. For some $i \leq n$, if $\rho_i = \rho_{i-1}$, then let $\rho := \rho_i$.

From M and $\mathcal{A}$, the procedure **Mini** shown in Figure 4 computes a dfma# $\mathcal{A}'$ with an initial assignment of length $n' \leq n$ as follows: A relation $\equiv_l$ used in **Mini** is defined;

Definition 6 *Let $W_p = \{w \in \Sigma^* | \delta(q_0,w) = p\}$. Then, $p \equiv_l q \Leftrightarrow^{def}$ there exists a one-to-one and onto mapping $l : \Sigma \mapsto \Sigma$ such that $W_q = \cup_{w \in W_p}\{l(w)\}$.*

For each $p \in S$ such that $\rho(p) = n$, if there exists $q' \in S$ such that $(p,n,q),(p,j,q') \in \mu$ for some $j < n$ and $q \equiv_l q'$, then let $\rho' = (\rho\backslash\{\rho(p) = n\}) \cup \{\rho(p) = j\}$, $\mu' = \mu\backslash\{(p,n,q)\}$ and $\mathrm{u}' = \#^{n-1}$. Remove $q \in S$ if $q_0 \not\vdash^* q$ or any $q_0 \vdash^* q$ is not defined with no transition of the form (p_i,n,p_j). Let S' be such a reduced set of states and $F' = S' \cap F$. For each $p \in S$ and $p' \in S'$ such that $p \equiv_l p'$, if, for each $a \in \Sigma$, there exists $a' \in \Sigma$ such that $\delta(p,a) \equiv_l \delta(p',a')$ and $\delta(p',a') \in S'$, then let $\mathcal{A} := \langle S', \mathrm{u}', q_0, \rho', \mu', F'\rangle$ and repeat the process. Otherwise output $\mathcal{A}$.

Let $\mathcal{A}$ be an actual hypothesis of $DFMA$ for a target $\mathcal{A}^*$. If $L(\mathcal{A}) \neq L(\mathcal{A}^*)$, then a counterexample is returned and a hypothesis is computed using membership and equivalence queries. Once 'yes' is returned, by Theorem 2, $DFMA$ outputs a correct hypothesis and halts.

```
Initialize a dfma# $\mathcal{A}$ such that $L(\mathcal{A}) = \emptyset$, $\Sigma = \emptyset$ and $E = \emptyset$.
begin
  E: make the equivalence query for $\mathcal{A}$.
     if the answer is 'yes', then
        output $\mathcal{A}$ and halt.
     otherwise let $w$ be the counterexample returned.
        set $E := E \cup \{w\}$ and $\Sigma := \Sigma \cup range(w)$
        $M := Table(\Sigma, E)$, $\mathcal{A} := Mini(M, Cons(M))$ and goto E. end
```

Figure 2: Algorithm **DFMA**

4.2 Correctness

Proposition 1 ***(Kaminski and Francez [9]).*** *Let $\mathcal{A} = \langle S, q_0, \mathrm{u}, \rho, \mu, F\rangle$ be a finite-memory automaton. Then, for each automorphism $l : \hat{\Sigma} \mapsto \hat{\Sigma}$, we have that $l(L(\mathcal{A})) = \cup_{w \in L(\mathcal{A})} l(w) = L(\mathcal{A}_{q_0,l(u)})$, where $A_{q_0,l(u)} =< S, q_0, l(u), \rho, \mu, F >$.*

From this proposition, we have that, for each dfma# $\mathcal{A}$ and each automorphism $l : \Sigma \mapsto \Sigma$, $l(L(\mathcal{A})) = L(\mathcal{A})$. Let $M = \langle S, \Sigma, q_0, \delta, F\rangle$ be a minimum dfa such that $L(M) = \Sigma^* \cap L$ for some $L \in DFMA^{\#}$. Then, for each one-to-one and onto mapping $l : \Sigma \mapsto \Sigma$, we have that $l(L(M)) = L(M)$.

Lemma 4 *If $p \not\equiv_l q$, then $l(W_p) \cap W_q = \emptyset$, where $l(W_p) = \cup_{w \in W_p} l(w)$.*

```
Input: a dfa $M = \langle S^M, \Sigma, q_0^M, \delta, F^M \rangle$, where $\Sigma = \{a_1, a_2, \ldots, a_n\}$.
Initialize a dfma# $\mathcal{A} = \langle S, \mathfrak{u}, q_0, \rho, \mu, F \rangle$ such that $S := S^M$, $\mathfrak{u} = \#^n$, $q_0 = q_0^M$, $\rho = \emptyset$
and for all $p, q \in S$ and $i \in \{1, 2, \ldots, n\}$, $(p, i, q) \in \mu$ iff $\delta(p, a_i) = q$.
begin
  for each($i \in \{1, 2, \ldots, n\}$ and $p \in S$)do
    if($\rho(p)$ is not defined and $\exists w \in (\Sigma \backslash \{a_i\})^*$ s.t $\delta(q_0, w) = p$)then
      $\rho(p) = i$.
  output $A$.
end
```

Figure 3: Procedure **Cons**

Proof. Assume the contrary, that is, there exist $w_1, w_2 \in W_p$ such that $l(w_1) \in W_q$ and $l(w_2) \in W_{q'}$ for some $l : \Sigma \mapsto \Sigma$, where $q \neq q'$. For each $w \in \Sigma^*$, $\delta(q_0, w_1 w) \in F$ iff $\delta(q_0, w_2 w) \in F$. Since $l(L(M)) = L(M)$, we have that, for each $w \in \Sigma^*$, $\delta(q_0, l(w_1 w)) \in F$ iff $\delta(q_0, l(w_2 w)) \in F$, that is, $\delta(q, l(w)) \in F$ iff $\delta(q', l(w)) \in F$. Since l is a one-to-one and onto mapping, $\cup_{w \in \Sigma^*} l(w) = \Sigma^*$. Thus, for each $w \in \Sigma^*$, $\delta(q, w) \in F$ iff $\delta(q', w) \in F$. Note that M is a minimum dfa. Contradiction. $\square$

Lemma 5 *Let $p \equiv_l q$. Then, for each $a \in \Sigma$, if $\delta(p, a) = p'$ and $\delta(q, l(a)) = q'$, then $p' \equiv_l q'$.*

Proof. From Lemma 4, it is sufficient to show that there exists at least one string $\alpha \in W_{p'}$ such that $l(\alpha) \in W_{q'}$. Let $w \in W_p$ and $w' \in W_q$. Then, for each $a \in \Sigma$, $wa \in W_{p'}$ and $w' l(a) = l(wa) \in W_{q'}$. $\square$

Let $\mathcal{A} = \langle S, \mathfrak{u}, q_0, \rho, \mu, F \rangle$ be a dfma# computed by **Cons** on input M. Consider a state $\mathcal{A}(w)$ for $w \in \Sigma^*$. Also, $\mathcal{A}(w)$ is a state of M. Then, if, for each $w \in \Sigma'^*$, $\delta(q_0, w) \equiv_l \mathcal{A}(w)$, then $L(M) = \Sigma^* \cap L(A)$, because $l(L(M)) = L(M)$ for each $l : \Sigma \mapsto \Sigma$.

Lemma 6 $L(M) = \Sigma^* \cap L(\mathcal{A})$.

Proof. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ and $\lambda = \{a_i \mapsto i | 1 \leq i \leq n\}$. For $w = w_1 w_2 \cdots w_m \in \Sigma^m$, let $(q_0, \mathfrak{u}^{\mathcal{A}}) \vdash^{w_1} (p_1, \mathfrak{u}_1) \vdash^{w_2} \cdots (p_{m-1}, \mathfrak{u}_{m-1}) \vdash^{w_m} (p_m, \mathfrak{u}_m)$, where $(p_{i-1}, \mathfrak{u}_{i-1}) \vdash^{w_i} (p_{i-1}, \mathfrak{u}_{i-1})$ for $(p_{i-1}, j_i, p_i) \in \mu$ $(1 \leq i \leq m)$. Then, define a string $w_\lambda = \lambda^{-1}(j_1 j_2 \cdots j_m) \in \Sigma^m$. We first show that for each $w \in \Sigma^+$, $\mathcal{A}(w) = \mathcal{A}(w_\lambda)$. It is true on each $w \in \Sigma$. Then, assume $\mathcal{A}(w) = \mathcal{A}(w_\lambda)$ on each $w \in \Sigma^n$. Let $(q_0, \mathfrak{u}^{\mathcal{A}}) \vdash^{\lambda^{-1}(j_1)} (p_1, \mathfrak{u}'_1) \vdash^{\lambda^{-1}(j_2)} \cdots (p_{m-1}, \mathfrak{u}'_{m-1}) \vdash^{\lambda^{-1}(j_m)} (p_m, \mathfrak{u}'_m)$. Let $a = u_w[k]$ for some $1 \leq k \leq |u^{\mathcal{A}}|$. Then, there exists $1 \leq i \leq m$ such that $(p_{i-1}, \mathfrak{u}_{i_1}) \vdash^a (p_i, \mathfrak{u}_i)$ for $(p_{i-1}, k, p_i) \in \mu$. By the induction hypothesis, $(p_{i-1}, \mathfrak{u}'_{i_1}) \vdash^{\lambda^{-1}(k)} (p_i, \mathfrak{u}'_i)$ for $(p_{i-1}, k, p_i) \in \mu$. Thus, $(p_m, \mathfrak{u}_m) \vdash^a (p_{m+1}, \mathfrak{u}_{m+1})$ for $(p_m, k, p_{m+1}) \in \mu$ implies $(p_m, \mathfrak{u}'_m) \vdash^{\lambda^{-1}(k)} (p_{m+1}, \mathfrak{u}'_{m+1})$ for $(p_m, k, p_{m+1}) \in \mu$. Let $a \notin [u_w]$, $\rho(p_m) = k$ and $(p_m, \mathfrak{u}_m) \vdash^a (p_{m+1}, \mathfrak{u}_{m+1})$ for $(p_m, k, p_{m+1}) \in \mu$. Thus, for each $a \in \Sigma$, $\mathcal{A}(wa) = \mathcal{A}((wa)_\lambda)$.

Next, we show that for each $w \in \Sigma^*$, $\delta(q_0, w) \equiv_l \mathcal{A}(w)$. On basis, $\delta(q_0, \varepsilon) = q_0 \equiv_l q_0 = \mathcal{A}(\varepsilon)$. Assume the induction hypothesis on some $w \in \Sigma^n$, that is, $\delta(q_0, w) = p$,

```
Input: a dfa $M = \langle S, \Sigma, q_0, \delta, F\rangle$, and a dfma# Cons($M$).
Let Cons($M$) $= \mathcal{A} = \langle S, \mathrm{u}, q_0, \rho, \mu, F\rangle$ and $n := |u|$
begin
  for each($p \in S$ s.t $\rho(p) = n$)do
    if($\exists j < n$ s.t $(p, n, q), (p, j, q') \in \mu$ and $q \equiv_l q'$)then
      $\rho' := \rho\backslash\{\rho(p) = n\} \cup \{\rho(p) = j\}$,  $\mathrm{u}' := \#^{n-1}$,  $\mu' := \mu\backslash\{(p, n, q)\}$,
      $S' := \{p \in S | q_0 \vdash^* p$ with no $(p_i, n, p_j) \in \mu\}$ and $F' := S' \cap F$.
      for each($p \in S$ and $p' \in S'$ s.t $p \equiv_l p'$)do
        if($\forall\delta(p, a) \in S, \exists\delta(p', a') \in S'$ s.t $\delta(p, a) \equiv_l \delta(p', a')$)then
          $A := \langle S', \mathrm{u}', q_0, \rho', \mu', F'\rangle$ and $n := n - 1$.
        else goto O.
    else goto O.
  O: output $\mathcal{A}$.
end
```

Figure 4: Procedure **Mini**

$\mathcal{A}(w) = p'$ and $p \equiv_l p'$. Since $\mathcal{A}(w) = \mathcal{A}(w_\lambda) = \delta(q_0, w_\lambda) = p'$, there exists $l^{-1}(w_\lambda) \in \Sigma^n$ such that $\delta(q_0, l^{-1}(w_\lambda)) = p$. Let $(wa)_\lambda = w_\lambda a_\lambda$. Since $a = u_w[i]$ iff $a_\lambda = u_\lambda[i]$, $\mathcal{A}(wa) = \mathcal{A}(w_\lambda a_\lambda)$. If $a \in [l^{-1}(w_\lambda)]$, then $l(l^{-1}(w_\lambda)a) = w_\lambda a_\lambda = (wa)_\lambda$. Thus, $\delta(q_0, wa) = \delta(q_0, l^{-1}(w_\lambda)a) \equiv_l \mathcal{A}((wa)_\lambda) = \mathcal{A}(wa)$. If $a \notin [l^{-1}(w_\lambda)]$, then $a_\lambda \notin [w_\lambda]$. Thus, there exists $l' = (l\backslash\{a \mapsto b\}) \cup \{a \mapsto a_\lambda\}$ for some $b \in \Sigma$ such that $l'(l^{-1}(w_\lambda a)) = w_\lambda a_\lambda = (wa)_\lambda$. Thus, $\delta(q_0, wa) = \delta(q_0, l^{-1}(w_\lambda)a) \equiv_{l'} \mathcal{A}(w_\lambda a_\lambda) = \mathcal{A}(wa)$. Hence, we have that for each $w \in \Sigma^*$, $\delta(q_0, w) \equiv_l \mathcal{A}(w)$. It follows that $L(M) = \Sigma^* \cap L(\mathcal{A})$. □

Similarly, we show that the relation $\equiv_l$ is also preserved on each dfma# computed by **Mini**. Let $\mathcal{B} = \langle S^{\mathcal{B}}, \mathrm{u}^{\mathcal{B}}, q_0, \rho^{\mathcal{B}}, \mu^{\mathcal{B}}, F^{\mathcal{B}}\rangle$ be an output of **Mini** on input M and $\mathcal{A}$.

Lemma 7 $\Sigma^* \cap L(\mathcal{B}) = \Sigma^* \cap L(\mathcal{A})$ *and the length of* $\mathrm{u}^{\mathcal{B}}$ *is minimum.*

Proof. We can assume that $S^{\mathcal{B}} \subseteq S^{\mathcal{A}}$, $\mu^{\mathcal{B}} \subseteq \mu^{\mathcal{A}}$, $F^{\mathcal{B}} \subseteq F^{\mathcal{A}}$, $|u^{\mathcal{B}}| \leq |u\mathcal{A}|$ and $\rho^{\mathcal{B}}(p) = \rho^{\mathcal{A}}(p)$ if $\rho^{\mathcal{A}}(p) \leq |u^{\mathcal{B}}|$. Thus, for each $p \in S^{\mathcal{A}}$, there exists $p' \in S^{\mathcal{B}}$ such that $p \equiv_l p'$, and for each $1 \leq i \leq n$, there exists $1 \leq i' \leq n'$ such that $q \equiv_l q'$ for $(p, i, q) \in \mu^{\mathcal{A}}$ and $(p', i', q') \in \mu^{\mathcal{B}}$. For $w = w_1 w_2 \cdots w_n \in \Sigma^n$, let $(q_0, \mathrm{u}) \vdash^{w_1} (p_1, \mathrm{u}_1) \vdash^{w_2} \cdots (p_{n-1}, \mathrm{u}_{n-1}) \vdash^{w_n} (p_n, \mathrm{u}_n)$, where $(p_{i-1}, \mathrm{u}_{i-1}) \vdash^{w_i} (p_i, \mathrm{u}_i)$ for $(p_{i-1}, j_i, p_i) \in \mu^{\mathcal{A}}$. Let $\Sigma' = \{a_1, a_2, \ldots, a_{n'}\}$. Then, there exists $w' = w_1' w_2' \cdots w_n' \in \Sigma'^n$ such that $(q_0, \mathrm{u}) \vdash^{w_1'} (p_1', \mathrm{u}_1') \vdash^{w_2'} \cdots (p_{n-1}', \mathrm{u}_{n-1}') \vdash^{w_n'} (p_n', \mathrm{u}_n')$, where $p_i \equiv_l p_i'$, and $(p_{i-1}', \mathrm{u}_{i-1}') \vdash^{w_i'} (p_i', \mathrm{u}_i')$ for $(p_{i-1}', j_i', p_i') \in \mu^{\mathcal{A}}$ $(j_i' \leq n')$. Thus, $\mathcal{A}(w) \equiv_l \mathcal{A}(w')$. Since no transition in $\mu^{\mathcal{A}}\backslash\mu^{\mathcal{B}}$ is applied in the computation $(q_0, \mathrm{u}) \vdash^{w'} (p_n', \mathrm{u}_n')$, this is a computation of $\mathcal{B}$, too. Thus, $\mathcal{A}(w') = \mathcal{B}(w')$. Now, we recall Definition 4. Then, from the condition $||\Sigma|| \geq |n^{\mathcal{B}}|$, we have that $w' \in w_{\Sigma}^{\mathcal{B}}$. Thus, by using Lemma 2, $\mathcal{B}(w') = \mathcal{B}(w)$. Consequently, $\mathcal{A}(w) \equiv_l \mathcal{A}(w') = \mathcal{B}(w') = \mathcal{B}(w)$. Hence, $\Sigma^* \cap L(\mathcal{A}) = \Sigma^* \cap L(\mathcal{B})$.

Next, we show that the length of $\mathrm{u}^{\mathcal{B}}$ is minimum. Assume that there exists a dfma# $\mathcal{C}$ such that $\Sigma^* \cap L(\mathcal{C}) = \Sigma^* \cap L(\mathcal{B})$ and the length of the initial assignment of $\mathcal{C}$ is shorter than $|n^{\mathcal{B}}|$. Since $|u^{\mathcal{C}}| < ||\Sigma||$, for each $w \in \Sigma^*$ there exist $a, b \in \Sigma$ such that $\mathcal{C}(wa) = \mathcal{C}(wb)$. Since $L(M) = \Sigma^* \cap L(\mathcal{C})$, it follows that $\delta(q_0, wa) \equiv_l \delta(q_0, wb)$. For $\mathcal{B}$, there exists $w \in \Sigma^*$ such that for each $a, b \in \Sigma$, $\mathcal{B}(wa) \not\equiv_l \mathcal{B}(wb)$. Since for each $w \in \Sigma^*$, $\delta(q_0, w) \equiv_l \mathcal{B}(w)$, we have $\delta(q_0, wa) \not\equiv_l \delta(q_0, wb)$. Since M is a minimum dfa, by Lemma 4, either $\delta(q_0, wa) \equiv_l \delta(q_0, wb)$ or $\delta(q_0, wa) \not\equiv_l \delta(q_0, wb)$ holds. Contradiction. $\square$

It is easy to see that the procedure **Cons** terminates on each input dfa. The main part of the procedure **Mini** is to decide, for any two states of input dfa, whether or not the relation $\equiv_l$ holds. Let $M = \langle S, \Sigma, q_0, \delta, F\rangle$ be an input dfa such that $||S|| = n$. For $p \in S$, let $W_p^n = \{w \in W_p | |w| \leq n\}$. Then, by Lemma 4, $p \equiv_l q$ iff $l(W_p^n) = W_q^n$. Since Σ, W_p^n and W_q^n are finite, it is computable whether or not $p \equiv_l q$. Thus, the procedure **Mini** also terminates. From Theorem 1, Lemma 6 and 7, we have the following.

Theorem 3 *The algorithm* ***DFMA*** *eventually terminates and outputs a dfma# $\mathcal{A}$ such that $L = L(\mathcal{A})$ for each $L \in$DFMA$^\#$.*

4.3 Running time

Lemma 8 *Given a minimum dfa M such that $L(M) = \Sigma^* \cap L$ for some $L \in$DFMA$^\#$. Let n be the number of states of M and $k = ||\Sigma||$. Then, for all pair of states p and q of M, the time to compute whether or not $p \equiv_l q$ is $\mathcal{O}(k \cdot n^3)$.*

Proof. Let $M = \langle S, \Sigma, q_0, \delta, F\rangle$ and $||S|| = n$. Then, S is divided into some disjoint sets as follows; $S = S_0 \oplus S_1 \oplus \cdots \oplus S_m$, where $m \leq n$ and $S_i = \{p \in S | \forall w \in W_p, |w| \geq i\}$ for $0 \leq i \leq m$. This computation is $\mathcal{O}(n)$ time. By the definition of $\equiv_l$, if $p \in S_i$, $q \in S_j$ and $i \neq j$, then $p \not\equiv_l q$. By the definition of S_i, clearly $S_0 = \{q_0\}$. Assume that it is decided whether or not $p \equiv_l q$ for each $p, q \in S_i$ for some $0 \leq i \leq m-1$. Let $p, q \in S_{i+1}$. Define $S_p, S_q \subseteq S_i$ as $p' \in S_p$ (or $q' \in S_q$) iff there exists $a \in \Sigma'$ such that $\delta(p', a) = p$ (or $\delta(q', a') = q$).

Then we can check whether or not $p \equiv_l q$ as follows: Let $p' \equiv_l q'$. Then we know a pair of strings (w, w') such that $l(w) = w'$, $\delta(q_0, w) = p'$ and $\delta(q_0, w') = q'$. It follows that we can compute a partial function $l' \subseteq l$ such that $l'(w) = w'$.

If $a \mapsto a' \in l'$ or $a \mapsto b \notin l'$ for each $b \in \Sigma$, then there exists $l'' = l' \cup \{a \mapsto a'\}$ such that $l''(wa) = w'a'$, $\delta(q_0, wa) = p$ and $\delta(q_0, w'a') = q$. Thus $p \equiv_l q$.

Otherwise for each $w \in \Sigma^*$ such that $\delta(q_0, w) = p'$ and for each $w' \in \Sigma^*$ such that $\delta(q_0, w') = q'$, we have that $l(wa) \neq w'a'$. For each $p', q' \in S_i$ such that $p' \equiv_l q'$ and for each $a, a' \in \Sigma$ such that $\delta(p', a) = p$ and $\delta(q', a') = q$, if $l(wa) \neq w'a'$, then clearly $p \not\equiv_l q$. Thus it can be decided whether or not $p \equiv_l q$ for each $p, q \in S_{i+1}$.

To decide $p \equiv_l q$, it takes at most $c_1 \cdot k \cdot ||S_i||$ steps for some constant c_1. Thus for each $p, q \in S_{i+1}$, it takes at most $c_2 \cdot k \cdot ||S_i|| \cdot ||S_{i+1}||^2$ steps for some constant c_2. Thus for all $p, q \in S$, it takes at most $c_3 \cdot k \cdot \sum_{i=0}^{m-1} ||S_i|| \cdot ||S_{i+1}||^2$ steps for some constant c_3. Since $\sum_{i=0}^{m-1} ||S_i|| = n$, the time is $\mathcal{O}(k \cdot n^3)$. $\square$

By Angluin's result, the running time of **Table** is $\mathcal{O}(m^2n^2 + mn^3)$, where n is the number of states of a minimum dfa and m is the length of a longest counterexample. The running time of **Cons** is linear in n. By Lemma 8, the running time of **Mini** is $\mathcal{O}(n^3)$. The number of repetition of **Table** and **Cons** is at most $2r$ times, respectively, where r is the length of a shortest initial assignment in dfma#'s for a target. Thus, the total running time is $\mathcal{O}((1 + 2 + \cdots + 2r)(m^2n^2 + mn^3 + rn^3)) = \mathcal{O}((rmn)^2 + r^2mn^3 + (rn)^3)$. Finally, we have the following result.

Theorem 4 *The class of simple deterministic finite-memory automata is learnable using membership and equivalence queries in polynomial time in r, n and m, where r is the length of a shortest initial assignment of dfma#'s for a target, n is the number of states of a minimum dfa consistent with a target over a finite alphabet of cardinality r, and m is the length of a longest counterexample returned so far.*

5 Concluding remarks

The class of simple deterministic finite-memory automata, denoted by dfma#'s, was defined in this paper. We have discussed the learnability of dfma#'s via membership and equivalence queries. In particular, we concluded that equivalence queries for dfma#'s are reasonable as well as membership queries for dfma#'s by proving that the equivalence of any two dfma#'s is decidable. Using this result, we provided a learning algorithm allowed to make polynomially many membership and equivalence queries. The main result of this paper was followed from the analysis of correctness and running time of the algorithm.

Almost results of this paper heavily depend on the closure property: for every dfma# $\mathcal{A}$ and automorphism $\ell : \hat{\Sigma} \mapsto \hat{\Sigma}$, it holds that $\ell(L(A)) = L(A)$. This closure property is concerned with initial assignments of dfma#'s, that include only the symbol $\#$. However, an initial assignment of a finite memory-automata contains some alphabet symbols. When a (deterministic) finite memory-automaton reads an input symbol contained in its initial assignment, it may pay special attention for the symbol in comparison with other symbols not contained initially. In this sense, every symbol is fairly judged on dfma#'s but not on dfma's. On the other hand, there is no difference between definitions of dfma's and dfma#'s except their initial assignments. Then, it seems that the problem of learning whole class of dfma's is reduced to that of identifying initial assignments of dfma's. Finally, the author summarize an idea to identify an initial assignment. Let $\mathcal{A}^*$ be a target dfma with an initial assignment $\mathrm{u} \in (\hat{\Sigma} \cup \{\#\})^n$ and $\mathcal{B}^*$ a dfma# obtained by changing u of $\mathcal{A}^*$ by $\#^n$. Without loss of generality, we can assume that an initial assignment of length n is of the form $\mathrm{u} = x_1 x_2 \cdots x_n$ such that $x_i \neq x_j$ if $i \neq j$, that is, every symbol in $\hat{\Sigma}$ occurs in u at most one time (See [9]). The first step is to identify an assignment of length n $\mathrm{u}' = y_1 y_2 \cdots y_n$ such that $y_i \neq y_j$ if $i \neq j$ and for each $1 \leq i \leq n$, $y_i = \#$ iff $x_i = \#$. Let $X = [\mathrm{u}]$ and $Y = [\mathrm{u}']$. Then, there exists one-to-one mapping φ: $X \mapsto Y$ such that $\varphi(\mathrm{u}) = \mathrm{u}'$. Let $\mathcal{B}$ be a dfma obtained by changing $\#^n$ of $\mathcal{B}^*$ by u'. From Proposition 1, for each automorphism $\ell : \hat{\Sigma} \mapsto \hat{\Sigma}$ such that $\varphi \subseteq \ell$, we have that $\ell(L(\mathcal{A}^*)) = L(\mathcal{B})$. The next step is to find a string in the symmetric difference of $L(\mathcal{A}^*)$ and $L(\mathcal{B})$. That is, we need to show that equivalence queries are reasonable on general dfma's, too. If a counterexample w is given, then by checking if $\varphi'(w) \in L(\mathcal{A}^*)$ and $\varphi'(w) \in L(\mathcal{B})$ for each $\varphi' : \Sigma \mapsto \Sigma$, we can decide the mapping φ. Then, $L(\mathcal{A}^*) = L(\mathcal{B}^{\varphi})$, where $\mathcal{B}^{\varphi}$ is obtained by changing u' of $\mathcal{B}$ by $\varphi^{-1}(\mathrm{u}')$. In future, the author would like to study the problem.

References

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

[2] D. Angluin. *Queries and concept learning.* Machine Learning, 2:319-342, 1988.

[3] A. Burago. *Learning structurally reversible context-free grammars from queries and counterexamples in polynomial time.* Proceedings of the 7th Workshop on Computational Learning Theory, New Brunswick, USA, pp.140-146, 1994.

[4] F. Bergadano and S. Varricchio. *Learning Behaviors of Automata from Shortest Counterexamples.* Proceedings of the 2nd European Conference on Computational Learning Theory, Barcelona, Spain, pp.380-391, 1995.

[5] W.I. Gasarch and C.H. Smith. Learning via queries. Journal of the ACM, 39(3):649-674, 1992.

[6] R. Gavaldà. *On the power of equivalence queries.* Proceedings of the 1st European Conference on Computational Learning Theory, Royal Holloway University, London, 1993.

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[8] H. Ishizaka. *Learning Simple deterministic Languages.* Proceedings of the 2nd Workshop on Computational Learning Theory, Morgan Kaufmann Publishers, Inc., San Mateo, CA, pp.162-174, 1989.

[9] M. Kaminski and N. Francez. *Finite-memory automata.* Theoretical Computer Science, 134:329-363, 1994.

[10] Y. Sakakibara. *Learning context-free grammars from structural data in polynomial time.* Theoretical Computer Science, pp.223-242, 1990.

[11] C. Sammut and R. Banerji, Learning concepts by asking questions, In R.S. Michalski, J.G. Carbonell and T.M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach*, Vol. 2, 1986, Morgan Kaufmann, San Mateo, Ca.

[12] E. Shapiro, A general incremental algorithm that infers theories from facts, *in* "Proc. 7th International Joint Conference on Artificial Intelligence," pp. 446 – 451, 1981, Morgan Kaufmann, San Mateo, Ca.

[13] E. Shapiro, Algorithmic Program Diagnosis, *in* "Proc. 9th ACM Symposium on Principles of Programming Languages," pp. 299 – 308, 1982, ACM Press

[14] E. Shapiro, *Algorithmic program debugging*, Cambridge, MA: MIT Press, 1983.

[15] H. U. Simon. *Learning decision lists and trees with equivalence-queries.* Proceedings of the 2nd European Conference on Computational Learning Theory, Barcelona, Spain, 1995.

[16] O. Watanabe. *A formal study of learning via queries.* Proceedings of the 17th International Colloquium on Automata, Languages and Programming, pp.139-152, 1990.

[17] T. Yokomori. *Polynomial-Time Identification of Very Simple Grammars from Positive Data.* Proceedings of the 4th Workshop on Computational Learning Theory, Santa Cruz, CA, pp.213-227, 1991.