Machine Learning, 19, 153–179 (1995)
© 1995 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

On Polynomial-Time Learnability in the Limit of Strictly Deterministic Automata

TAKASHI YOKOMORI yokomori@cs.uec.ac.jp

Department of Computer Science and Information Mathematics, University of Electro-Communications, 1-5-1 Chofugaoka, Chofu, Tokyo, 182 Japan

Editor: Leonard Pitt

Abstract. This paper deals with the polynomial-time learnability of a language class in the limit from positive data, and discusses the learning problem of a subclass of deterministic finite automata (DFAs), called *strictly deterministic automata* (SDAs), in the framework of learning in the limit from positive data. We first discuss the difficulty of Pitt's definition in the framework of learning in the limit from positive data, by showing that any class of languages with an infinite descending chain property is not polynomial-time learnable in the limit from positive data. We then propose new definitions for polynomial-time learnability in the limit from positive data. We show in our new definitions that the class of SDAs is iteratively, consistently polynomial-time learnable in the limit from positive data. In particular, we present a learning algorithm that learns any SDA M in the limit from positive data, satisfying the properties that (*i*) the time for updating a conjecture is at most $O(\ell m)$, (*ii*) the number of implicit prediction errors is at most $O(\ell n)$, where ℓ is the maximum length of all positive data provided, m is the alphabet size of M and n is the size of M, (*iii*) each conjecture is computed from only the previous conjecture and the current example, and (*iv*) at any stage the conjecture is consistent with the sample set seen so far. This is in marked contrast to the fact that the class of DFAs is neither learnable in the limit from positive data nor polynomial-time learnable in the limit.

Keywords: polynomial-time learnability in the limit, iterative learning, strictly deterministic automata

1. Introduction

In the study of inductive learning of formal languages, Gold (1967) showed that any class of languages containing all finite sets and at least one infinite set (which is called a *superfinite* class) is not learnable in the limit from positive data (that is, from the examples in the target language). This fact was shocking in the sense that even the class of regular languages is not learnable in the limit from positive data. Angluin (1980) has given several conditions for a class of languages to be learnable in the limit from positive data, and presented some examples of learnable classes. She has also proposed subclasses of regular languages called k-reversible languages for each non-negative integer k and shown these classes are learnable in the limit from positive data with a polynomial-time algorithm for updating conjectures (Angluin, 1982). Motivated by a question posed by Angluin, however, one natural question has been recently recognized as significant: In what sense should we analyse the time complexity of an "in-the-limit" algorithm ? Because there are a great variety of possible definitions for *polynomial-time learnability* *in the limit.* Among others, Pitt has proposed the following definition for polynomial-time learnability in the limit from positive and negative data (Pitt, 1989). Informally, a class of concepts (or representations) $\mathcal{M}$ is *polynomial-time learnable in the limit* if and only if there is an algorithm $\mathcal{A}$ which, given a sequence of strings accepted by M in $\mathcal{M}$, learns an equivalent M' in $\mathcal{M}$ in the limit, with the property that there exist polynomials p and q such that the time for updating a conjecture is at most $p(n, N)$ and the number of times $\mathcal{A}$ makes a wrong conjecture is at most $q(n)$, where n is the size of M, and N is the sum of lengths of data provided. It turned out, however, that this definition was too strong to have positive results for non-trivial language classes. For example, a recent result shows that even the class of all finite languages or a subclass of regular languages called zero-reversible languages is not polynomial-time learnable in the limit (Angluin, 1990; Pitt, 1989). In fact, to the author's knowledge, no non-trivial class of languages has been yet proven polynomial-time learnable in the limit in this definition. It is beyond question that the polynomial-time learnability in the limit (in particular, from positive data) is of great significance from not only the theoretical but also the practical point of view.

In our previous paper (Tanida & Yokomori, 1992), we have introduced a subclass of DFAs called *strictly deterministic automata* (SDAs) and shown that the class is learnable in the limit from positive data requiring polynomial time for updating conjectures. An SDA is intuitively a state transition graph in which the set of labels W for edges is a finite subset of strings over an alphabet Σ. For each symbol a in Σ there is at most one w in W starting with a. The motivation for introducing the notion of an SDA is explained as follows. Consider the following sentences :

"you feel very happy"
"she is very very happy"
"she came long long ago"

These are recognized by, for instance, the acceptor M in Figure 1. We observe that all edges of M are labeled by words that all start with distinct letters, and that double occurrences of an identical word (say, "very" or "long") ensure the existence of a self-loop in the transition graph. These features lead to a language class of specific sentences with which we are concerned in the present paper. This notion is also motivated by the work on Szilard languages of linear grammars (Mäkinen, 1990). The class of Szilard languages of linear grammars is a proper subclass of the class of zero-reversible languages, while our class of languages properly includes the class of Szilard languages of linear grammars, but is incomparable to the class of zero-reversible languages.

In this article, we first discuss the difficulty of Pitt's definition in the framework of learning in the limit from positive data, by showing that any class of languages with an infinite descending chain property is not polynomial-time learnable in the limit from positive data. Since most of the existing language classes satisfy the property, this result strongly encourages us to propose new definitions for polynomial-time learnability in the

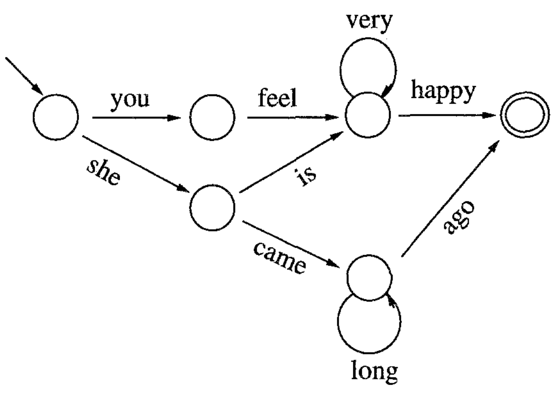

Figure 1. A Simple Natural Language Recognizer

limit from positive data. In fact, we propose two new definitions for polynomial-time learnability in the limit from positive data by making modifications to a straightforwardly restricted version of Pitt's definition for the case of positive data only.

Then, we make a visit to the class of SDAs again to generalize and strengthen the learnability results of SDAs in the new definition. We show that the class of SDAs is polynomial-time learnable in the limit from positive data, by presenting an improved version of the learning algorithm given in Tanida and Yokomori (1992). This result is in marked contrast to the fact that the class of DFAs is neither learnable in the limit from positive data nor polynomial-time learnable in the limit. As corollaries, it follows that the class of SDAs is learnable in polynomial time via equivalence queries only and it is also PAC-learnable in polynomial time from positive examples only. The main results in this paper extend and supplement those in not only Tanida and Yokomori (1992) but also Mäkinen (1990). Further, we discuss an interesting feature of our learning algorithm, namely, *iterative consistency* which enables the algorithm to be efficient in space as well.

2. Definitions

2.1. Basic Definitions and Notations

We assume the reader to be familiar with the basic notions in automata and formal language theory. For the notions and notations not stated here, see, e.g., Hopcroft and Ullman (1979). An *alphabet* Σ is a finite set of symbols. For any finite set S of finite-length strings over Σ, we denote by S^* (respectively, S^+) the set of all finite-length strings obtained by concatenating zero (one, resp.) or more elements of S, where the concatenation of strings u and v is simply denoted by uv. In particular, Σ^* denotes the set of all finite-length strings over Σ. The string of length 0 (the empty string) is denoted by λ. By $len(w)$ we denote the length of a string w. A *language* over Σ is any subset L of Σ^*. The cardinality of a set S is denoted by $|S|$. In a directed graph, called a *transition graph*, an edge from node p to node q labeled u is denoted by (p, u, q) and is called a *transition*. A string $u(\in \Sigma^+)$ is called the *label* of (p, u, q). For a language $L \subseteq \Sigma^*$ and $x \in \Sigma^*$, let $x\backslash L = \{y | xy \in L\}$.

2.2. Polynomial-time Learning in the Limit from Positive Data — Difficulty and New Definitions

Let Σ be fixed and let $\mathcal{M}$ be the class of acceptors to be learned, where each $M \in \mathcal{M}$ specifies a language L over Σ, denoted by $L = L(M)$. Let L be a language such that $L = L(M)$ for some M in $\mathcal{M}$. A string in L is called a *positive example* of L and positive examples are often called *positive data.* A *positive presentation* of the language L is any infinite sequence of strings such that every string $w \in L$ occurs at least once in the sequence, and no other strings occur in the sequence. In our model of learning, a learning algorithm $\mathcal{A}$ takes, as an input, an arbitrary positive presentation of L : $w_1, w_2, \cdots$, and outputs an infinite sequence of acceptors (conjectures): $M_1, M_2, \cdots$. An algorithm $\mathcal{A}$ is said to *learn an acceptor M in the limit from positive data* if and only if for any positive presentation of $L(M)$ the infinite sequence of acceptors in $\mathcal{M}$ produced by $\mathcal{A}$ satisfies the property that there exists M' in $\mathcal{M}$ such that for all sufficiently large i, the i-th conjecture (acceptor) M_i is identical to M' and $L(M') = L(M)$. A class of acceptors $\mathcal{M}$ is *learnable in the limit from positive data* if and only if there exists an algorithm $\mathcal{A}$ that, for any M in $\mathcal{M}$, learns M in the limit from positive data. For each $i \geq 1$, let w_i be the i-th example provided and M_i be the i-th conjectured acceptor produced by an algorithm $\mathcal{A}$. Then, $\mathcal{A}$ is said to make an *implicit error of prediction* at the i-th step if and only if the conjecture M_i fails to accept the $(i+1)$-st example w_{i+1}.

From the original Pitt's definition (Pitt, 1989) for polynomial-time learnability in the limit, one may propose the following definition for the case of positive data only.

Definition 1 A class $\mathcal{M}$ is *polynomial-time learnable in the limit from positive data* if and only if (1) $\mathcal{M}$ is learnable in the limit from positive data, and (2) the learning algorithm $\mathcal{A}$ in (1) satisfies the property that there exist polynomials p, q such that for any n, for any M of size n, and for any positive presentation of $L(M)$, the time used by $\mathcal{A}$ between receiving the i-th example w_i and outputting the i-th conjectured acceptor M_i is at most $p(n, l_1 + \cdots + l_i)$, and the number of implicit errors of prediction made by $\mathcal{A}$ is at most $q(n)$, where $l_j = len(w_j)(j = 1, 2, ..., i)$.

However, no non-trivial class of acceptors (or languages) has been yet proven polynomia time learnable in the limit in this definition. (In fact, it has been proved that even the class of all DFAs recognizing only finite languages is not polynomial-time learnable in the limit (Angluin, 1990). On the other hand, the class of DFAs is polynomial-time learnable in the limit with the help of membership queries (Angluin, 1987).)

Moreover, we can show the following negative result in this definition.

THEOREM 1 *Let $\mathcal{M}$ be any class of acceptors with the property that there exists an infinite sequence of acceptors $M_0, M_1, M_2, \cdots$ such that $L(M_0) \supset L(M_1) \supset L(M_2) \cdots$, that is, the sequence forms an infinite descending chain with respect to the inclusion relation. Then, $\mathcal{M}$ cannot be learned according to Definition 1.*

Proof: Suppose that there exists an infinite descending chain : $L(M_0)$, $L(M_1)$, $L(M_2), \cdots$, where for each $i \geq 0$ $L(M_i)$ properly contains $L(M_{i+1})$. Let $\mathcal{A}$ be any algorithm that learns every M in $\mathcal{M}$ in the limit from positive data. We show that for every positive integer k, we can construct a positive presentation of $L(M_0)$ on which $\mathcal{A}$ must make at least k implicit errors of prediction.

Begin by enumerating the elements of $L(M_k)$ and providing them as input to $\mathcal{A}$ until the output sequence of conjectures of $\mathcal{A}$ converges to an acceptor equivalent to M_k. This must happen in finite time, since $\mathcal{A}$ learns every M in $\mathcal{M}$ in the limit from positive data. When this happens, choose any element of $L(M_{k-1}) - L(M_k)$ as the next example in the positive presentation. On this element, $\mathcal{A}$ makes an implicit error of prediction. Then, we continue the construction of positive presentation by enumerating the elements of $L(M_{k-1})$ until the output sequence of $\mathcal{A}$ converges to an acceptor equivalent to M_{k-1}. At this point, choose any element of $L(M_{k-2}) - L(M_{k-1})$ as the next example in the input sequence on which $\mathcal{A}$ makes an implicit error of prediction, and we continue with an enumeration of $L(M_{k-2})$.

Thus, if we continue in this way, then the output sequence of $\mathcal{A}$ converges to an acceptor equivalent to M_0 after having made at least k implicit errors of prediction.

Let n_0 be the size of M_0. Then, since n_0 is some fixed value, no function $f(n)$ has the property that $k \leq f(n_0)$ for all positive integers k. Thus, no algorithm that learns every M in $\mathcal{M}$ in the limit from positive data can make a number of implicit errors of prediction bounded by a polynomial in the size of the target acceptor. ■

Since most of the non-trivial classes of acceptors have the infinite descending chain property stated in the theorem, this implies that Definition 1, a straightforward modification of Pitt's definition, is too restrictive to obtain any positive result of polynomial-time learnability in the limit from positive data, leading us to new definitions.

We now propose the following two new definitions for polynomial-time learnability in the limit from positive data.

Definition 2 A class $\mathcal{M}$ is *polynomial-time learnable in the limit from positive data* if and only if (1) $\mathcal{M}$ is learnable in the limit from positive data, and (2) the learning algorithm $\mathcal{A}$ in (1) satisfies the property that there exist polynomials p, q such that for any n, for any M of size n, and for any positive presentation of $L(M)$, the time used by $\mathcal{A}$ between receiving the i-th example w_i and outputting the i-th conjectured acceptor M_i is at most $p(n, l_1 + \cdots + l_i)$, and the number of implicit errors of prediction made by $\mathcal{A}$ is at most $q(n, \ell)$, where $l_j = len(w_j)$ $(j = 1, 2, ..., i)$, ℓ is the maximum length of any positive example provided up to that point of the run of the learning algorithm, and the size of M is assumed to be appropriately defined.

Definition 3 A class $\mathcal{M}$ is *polynomial-time learnable in the limit from positive data in the strict sense* if and only if, in addition to all the conditions in Definition 2, the algorithm $\mathcal{A}$ also satisfies the property that for any target acceptor M in $\mathcal{M}$ and for any positive presentation of $L(M)$, each conjectured acceptor M_i from $\mathcal{A}$ accepts a subset of $L(M)$, i.e., it holds that for all $i \geq 1$, $L(M_i) \subseteq L(M)$ (*the subset requirement*).

In comparison to Pitt's definition for polynomial-time learnability in the limit (Pitt, 1989), there are two major differences among these definitions. That is, in the new definitions above, the algorithm is allowed to have the maximum length of all data supplied as a new parameter for a polynomial that bounds the number of implicit errors of prediction (in (2)), while Definition 3, in turn, requires the algorithm to confine its conjectures to subsets of the target languages. Note that the learnability in Definition 3 obviously implies the learnability in Definition 2.

In the next section, we will introduce a non-trivial subclass of DFAs and show the class is polynomial-time learnable in the limit from positive data in the sense of Definition 2. In fact, we prove even a stronger result in the sense of Definition 3 whose introduction is motivated by the fact that it has some implications to other types of learning paradigms, in particular, learning with equivalence queries (Angluin, 1987) and PAC-Learning (Valiant, 1984)(see Corollary 17).

2.3. *Strictly Deterministic Automata and Their Languages*

We introduced in Tanida and Yokomori (1992) the notion of strictly deterministic automata which was defined in two steps, that is, first by extending the notion of (usual) deterministic finite automata and then by making a certain restriction on them. In this article, we will reformalize the notion in an extended manner.

Definition 4 (Wood, 1987) An *extended deterministic finite automaton* (EDFA) M over Σ is a 5-tuple (Q, T, δ, p_0, F), where

Q is a finite, nonempty set of states,

$T (\subseteq \Sigma^+)$ is a finite set of strings,

$\delta : Q \times T \rightarrow Q$ is a state transition function (possibly) partially defined,

p_0 is the initial state in Q, and

F $(\subseteq Q)$ is a set of final states.

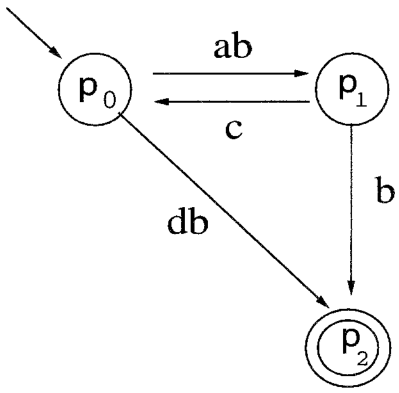

Figure 2. An SDA *M* accpeting *L(M)*

For convenience, we sometimes take δ as the set of transitions $\{(p, u, q) | \delta(p, u) = q\}$.

Definition 5 [Strictly Deterministic Automaton] Let $M = (Q, T, \delta, p_0, F)$ be an EDFA. Then, M is a *strictly deterministic automaton* (SDA) if and only if M satisfies the following:

(1) for any $u \in T$, there uniquely exists a pair (p, q) such that $(p, u, q) \in \delta$ (i.e., the uniqueness of labels).

(2) for any $u_1, u_2 \in T$, the first symbol of u_1 differs from that of u_2. (We say that T has the *strict prefix property*.)

A partial function δ is often extended from $Q \times T$ to $Q \times T^*$ in a usual manner:
for all $p \in Q, x \in T^*, u \in T$
$\delta(p, \lambda) = p$
$\delta(p, xu) = \delta(\delta(p, x), u)$.
The language accepted by M, denoted by $L(M)$, is defined by $\{w \in \Sigma^* | \delta(p_0, w) \in F\}$. A language L is called a *strictly regular language* (SRL) if $L = L(M)$ for some SDA M.
Note. Besides (1) and (2) above, the original definition of an SDA in Tanida and Yokomori (1992) requires an additional condition which disallows multi-edges in the transition graph of M.

Example **1** Let $\Sigma = \{a, b, c, d\}$. Consider an EDFA $M = (\{p_0, p_1, p_2\}, \{ab, b, c, db\}, \delta, p_0, \{p_2\})$, where $\delta = \{(p_0, ab, p_1), (p_0, db, p_2), (p_1, c, p_0), (p_1, b, p_2)\}$. Then, M is an SDA and $L(M)$ is the language expressed by the regular expression $(abc)^*(abb + db)$. The transition graph of M is shown in Figure 2. □

In what follows, we fix the alphabet Σ for SDAs or SRLs. Further, we assume that each SDA $M = (Q, T, \delta, p_0, F)$ has no *useless* states, where a state $q \in Q$ is said to be *useless* if it is not on any path from the initial state p_0 to a final state in F.

2.4. Characterization Results on SDAs

The next result is basic for all characterization results we will obtain in the sequel.

LEMMA 2 *Let $M = (Q, T, \delta, p_0, F)$ be an SDA, and p, p', q, q' be in Q, and $\alpha \in T^+$. If $\delta(p, \alpha) = q$ and $\delta(p', \alpha) = q'$, then $p = p'$ and $q = q'$.*

Proof: Let $\alpha = u_1 \cdots u_n$; $(u_i \in T, 1 \leq i \leq n)$. We will show by induction on n that if $\delta(p, \alpha) = q$ and $\delta(p', \alpha) = q'$, then $p = p'$ and $q = q'$. Assume that $n = 1$, then clearly the claim holds from the definition of SDA. Let us assume the claim holds for $n = k$ $(k \geq 1)$. Let $\beta = \alpha u_{k+1}$, where $\alpha = u_1 \cdots u_k$, and $u_{k+1} \in T$. Then,

$$\begin{aligned}\delta(p, \beta) &= \delta(p, u_1 \cdots u_k u_{k+1}) \\ &= \delta(\delta(p, \alpha), u_{k+1}) \\ &= \delta(r, u_{k+1}), \quad \text{where } r = \delta(p, \alpha) \\ &= q.\end{aligned}$$

Similarly, we have that $\delta(p', \beta) = \delta(r', u_{k+1}) = q'$, where $r' = \delta(p', \alpha)$. Then, from the inductive hypothesis, $p = p'$ and $r = r'$, and hence $q = q'$. Thus, the claim holds for $k = n + 1$, which completes the proof.

■

From Lemma 2, we have the following corollary.

COROLLARY 3 *Let $M = (Q, T, \delta, p_0, F)$ be an SDA, and let u_1, u_2 be in T^* and v be in T^+. If M accepts $u_1 v$ and $u_2 v$, then $\delta(p_0, u_1) = \delta(p_0, u_2)$.*

A regular language L is *zero-reversible* if and only if for $u_1, u_2 \in \Sigma^*$ and $v \in \Sigma^+$, whenever $u_1 v, u_2 v$ are in L, it holds that $u_1 \backslash L = u_2 \backslash L$ (Angluin, 1982). Corollary 3 means that an SRL is zero-reversible if it is restricted to T^*.

Example 2 Consider the strictly regular language $L = L(M)$ in Example 1. Clearly, L is not zero-reversible, because, for example, abb and db are both in L but $ab\backslash L \neq d\backslash L$. On the other hand, the language of the regular expression a^*ba^* is zero-reversible, but not strictly regular, which can be proved using Lemma 2. □

Since there is a language which is both strictly regular and zero-reversible (e.g., the language of the regular expression a^*b), the following theorem holds.

THEOREM 4 *The class of strictly regular languages is incomparable to but not disjoint with the class of zero-reversible languages.*

Using Lemma 2, it is easy to obtain the next lemma which gives an interesting property of SRLs.

LEMMA 5 *Let $M = (Q, T, \delta, p_0, F)$ be an arbitrary SDA. Then, for $x, z \in \Sigma^*$, $y \in \Sigma^+$, if xz, xyz and $xyyz$ are in $L(M)$, then for each $i \geq 0$, xy^iz is in $L(M)$.*

Proof: Let v be the longest common prefix of z and yz. Then, xv is the longest common prefix of xz and xyz, and xyv is the longest common prefix of xyz and $xyyz$. From the strict prefix property of an SDA, it is easy to see that if M accepts two strings x and y, then the longest common prefix of x and y reaches a state of M from q_0. Hence, let $p_1 = \delta(p_0, xv)$ and $p_2 = \delta(p_0, xyv)$.

Since v is the longest common prefix of z and yz, it is also the longest common prefix of z and yv. To see this, consider the following two cases. Case (1) $len(v) \leq len(y)$: Then, v is also clearly the longest common prefix of z and yv. Case (2) $len(v) > len(y)$: Let $z'(\in \Sigma^*)$ and $z''(\in \Sigma^+)$ be strings such that $z = vz'$ and $v = yz''$. Then, v can be also split as $(v =)z''v'$ for some $v' \in \Sigma^*$, where $z = z''v'z'$. Note that $z = vz' = (yz'')z'$, $yz = y(vz') = y(z''v')(z') = (yz'')v'z'$ and $yv = y(z''v') = (yz'')v'$. Since $v(= yz'')$ is the longest common prefix of z and yz, v is also the longest common prefix of z and yv. Thus, in either case, we conclude that v is also the longest common prefix of z and yv. This implies that there exists a string y' such that $yv = vy'$. Since y is non-empty, y' is non-empty. We now prove by induction on j the following : **[Claim]** for each $j \geq 0$, $y^jv = v(y')^j$. The case when $j = 0$ or $j = 1$ is trivially true. Now consider

$$\begin{aligned} y^jv &= yy^{j-1}v \\ &= yv(y')^{j-1} \text{ (by inductive hypothesis)} \\ &= vy'(y')^{j-1} \text{ (since } yv = vy'\text{)} \\ &= v(y')^j \quad \text{(End of the proof of Claim).} \end{aligned}$$

Let $z = vz'$. Then, since v is a prefix of z, it holds that

$$xyz = xyvz' = xvy'z' \text{ and } xyyz = xyyvz' = xyvy'z'.$$

Thus, as we have seen above, since $p_1 = \delta(p_0, xv)$ and $p_2 = \delta(p_0, xyv)$, we have that $\delta(p_1, y'z') = q_f \in F$ and $\delta(p_2, y'z') = q'_f \in F$. Since $y'z'$ is non-empty, from Lemma 2, we conclude that $p_1 = p_2$ (and $q_f = q'_f$). Since $xyv = xvy'$, this implies that $\delta(p_0, xv) = p_1 = p_2 = \delta(p_0, xvy')$ and therefore, $p_1 = \delta(p_1, y')$. Thus, for each $j \geq 0$, every string of the form $xv(y')^jz'$ is accepted by M, while it holds that

$$xv(y')^jz' = xy^jvz' = xy^jz.$$

Therefore, for each $j \geq 0$, M accepts every string of the form xy^jz.

■

3. Learning Strictly Deterministic Automata

3.1. *The Learning Algorithm LA*

We present a learning algorithm LA for SDAs which comprises four subprocedures named "UPDATE", "REFINE", "PARSE" and "CONSTRUCT"; we first give intuitive descriptions of these procedures.

Let T be a set of strings corresponding to edge labels of a conjectured SDA to be constructed by LA. We say that T *can parse a string* w iff w can be uniquely written as the concatenation of strings from the set T, i.e., if $w = x_1 \cdots x_m = x'_1 \cdots x'_n$ $(x_j, x'_k \in T; 1 \leq \forall j \leq m, 1 \leq \forall k \leq n)$, then $m = n$ and $x_j = x'_j$ $(1 \leq \forall j \leq m)$. It is easy to see that if w can be written as the concatenation of elements of T and T satisfies the strict prefix property, then w can be uniquely written. (Thus, provided that T has the strict prefix property, the fact that w can be written as the concatenation of elements of T immediately implies that T can parse w.)

Intuitively, the algorithm LA behaves as follows. The algorithm maintains a set T of edge labels that, at any time, is sufficient to parse all example strings seen so far. The goal of UPDATE is to update the set T so that the new example can be parsed, while maintaining the parsability of all previously seen examples. Suppose that the first positive example is $abba$, the set T consists only of the string $abba$ and clearly this is sufficient to parse itself. Now, if a new positive example is obtained, say $abaa$, then $T = \{abba\}$ is not sufficient to parse $abaa$. Since the edge labels of the target SDA M satisfy the strict prefix property, the edge label of M that begins with a must be a prefix of the longest common prefix of $abba$ and $abaa$, i.e., a prefix of ab, in order for M to parse both $abba$ and $abaa$. Thus, in T we replace $abba$ with ab. Now, the edge labels of M must parse the corresponding suffixes ba and aa of $abba$ and $abaa$, respectively, so UPDATE must ensure that both of these suffixes can be parsed. UPDATE maintains a queue of strings that need to be parsed in order for the final set T to be able to parse the new string and all previously parsable strings.

In general, the set T maintains a tentative set of edge labels of the SDA to be learned, and the elements of T always satisfy the strict prefix property. At any time, elements of T may be replaced, or they may be split into smaller pieces and new elements of T may be added.

After updating T with a new example, a procedure REFINE takes as input each transition from the currently conjectured SDA, and breaks it up into the new smaller parse strings of the updated set T, introducing a chain of new states if necessary. Then, a procedure PARSE adds into the current automaton a new chain of states (using the updated set T of edge labels) that can exactly parse the new example.

Finally, CONSTRUCT takes the refined automaton, along with the new chain of states for the parse of the new example, and then condenses it into a deterministic automaton using Lemma 2.4 to merge equivalent states and edges.

Let L be an SRL over $\Sigma = \{a_1, \ldots, a_m\}$ and $S_i = \{w_1, \cdots, w_i\}$ be the finite sample set of positive examples of L provided up to the i-th stage of the learning process.

(1) UPDATE

We describe how the set T is updated, when a new positive example w is provided. Let w_i be the i-th positive example provided at the i-th stage of the learning process and let $M_{i-1} = (Q_{i-1}, T_{i-1}, \delta_{i-1}, p_0, F_{i-1})$ be the conjectured SDA at the $(i-1)$-st stage, where the set of strings of edge labels T_i has the strict prefix property of Definition 5.

We now suppose that w_i is not consistent with M_{i-1}, that is, w_i is not in $L(M_{i-1})$.

The procedure UPDATE(T_{i-1}, w_i) takes as input T_{i-1} and w_i, and returns as output an updated set T_i such that (1) T_i can parse any string w in S_i (which includes the new string w_i), and (2) T_i has the strict prefix property.

In describing the procedure, the following notations will be used :

- firstchar(w) denotes the first symbol of w if $w \neq \lambda$
- lcp(u, v) denotes the longest common prefix of u and v.

The procedure UPDATE is given below.

[Correctness of UPDATE]

From the way T_{i-1} is updated, it is clear that UPDATE(T_{i-1}, w_i) always halts and returns an output set T_i. In order to prove the correctness of UPDATE, we must show that after the execution of UPDATE(T_{i-1}, w_i) the following properties hold :

(1) T_i can parse every string w in S_i, and

(2) T_i has the strict prefix property.

Let w be a string from Q. As a notation, $w \Rightarrow^x w'$ iff w' is a suffix of w obtained by making one application of an operation $x (\in \{(\mathrm{a}), (\mathrm{b}), (\mathrm{c})\})$ of the **case** statements in the **repeat** loop. That is,

$$w \Rightarrow^x w' \text{ iff } \begin{cases} x = (\mathrm{a}) \text{ and } w' = \lambda \text{ with } w \in T, \\ x = (\mathrm{b}) \text{ and } w = t_a w' \ \text{ with } t_a \in T, \\ x = (\mathrm{c}) \text{ and } w = x_a w' \text{ and } x_a = \mathrm{lcp}(w, t_a) \text{ with } x_a \in T. \end{cases}$$

The property (2) is shown by induction on i. For $i = 1$, $T_1 (= \{w_1\})$ trivially has the strict prefix property. To see that $T_i (= \text{UPDATE}(T_{i-1}, w_i))$ has the strict prefix property, from the inductive hypothesis, we have only to observe that the strict prefix property of T_{i-1} is preserved at any time through the operation of each **case** statement (a), (b) and (c) in the **repeat** loop, by construction.

Consider a sequence of operations : $w \Rightarrow^{x_1} w_1 \Rightarrow^{x_2} \cdots \Rightarrow^{x_{n-1}} w_{n-1} \Rightarrow^{x_n} w_n(= \lambda)$. In this case, a word $x_1 \cdots x_n$ $(\in \{(\mathrm{a}), (\mathrm{b}), (\mathrm{c})\}^+)$ is called the *associate word* of u.

```
Procedure UPDATE($T_{i-1}, w_i$) :
    begin
        let $T := T_{i-1}$ ;
        let $Q := \{w_i\}$ ;        /* $Q$ is a queue of strings to be parsed */
        let $w$ be the element of $Q$ removed from the top ;
                                    /* $w$ holds the current string being parsed */
        repeat (forever)
        if $w = \lambda$   then
                if $Q = \emptyset$   then  HALT outputting $T_i = T$
                else   let $w$ be the element of $Q$ removed from the top
        case
          (a) firstchar($w$) $\neq$firstchar($t$) for any $t \in T$ :
              let $T := T \cup \{w\}$ ;
              let $w := \lambda$ ;
          (b) $w = t_a w'$ (with $t_a \in T$) :
              let $w := w'$ ;
          (c) $w = x_a w'$ and $t_a = x_a t'_a$ (with $t_a \in T$, $t'_a \neq \lambda$ and $x_a = \mathrm{lcp}(w, t_a)$) :
              let $T := (T - \{t_a\}) \cup \{x_a\}$ ;        /* replace $t_a$ with $x_a$ */
              let $w := w'$ ;
              put $t'_a$ at the end of $Q$
        end repeat
    end
```

In order to prove the property (1) above, we first show the following claim :

[Claim] Every string w ever put into Q can be parsed by T_i.

Proof. Noting that every string w in Q is removed from the top of Q to be scanned, let α be the associate word of w. We show by induction on $n = len(\alpha)$ that w can be parsed by the current set T during the execution of UPDATE(T_{i-1}, w_i). For $n = 1$, by construction, exactly one of the following three cases occurs : (*i*) $T := T \cup \{w\}$ (in case (a)), (*ii*) $w = t_a$ with $t_a \in T$ (in case (b)), (*iii*) $w = x_a (= \mathrm{lcp}(w, t_a)) \in T$ (case (c)). Thus, in all cases, w can be parsed by T. Suppose inductively that any string in Q whose associate word has length less than n can be parsed by T, and consider $w \Rightarrow^{x} w_1 \Rightarrow^{\alpha'} \lambda$ (for some w_1), where $\alpha = x\alpha'$ is the associate word of w, $len(\alpha') = n - 1$ and $x \in \{(\mathrm{b}), (\mathrm{c})\}$. Then, by definition, either $w = t_a w_1$ with $t_a \in T$ or $w = x_a w_1$ with $x_a \in T$. By the inductive hypothesis, since w_1 can be parsed by T, we conclude that w can be parsed by T. T may be further updated by case (c) to finish scanning all strings in Q. In case (c), suppose that t'_a is put into Q, where

$w = x_a w'$, $t_a = x_a t'_a$ and t_a is replaced with x_a in T. But, as argued above, since t'_a is parsed by the current T, T can also parse t_a. Thus, we conclude that when UPDATE eventually terminates, T_i can parse every string w in Q. (End of the proof of Claim.)

We now prove the property (1) by induction on i. First, $T_1 = \{w_1\}$ can trivially parse every string in $S_1 (= \{w_1\})$. Thus, we may assume as an induction hypothesis that T_{i-1} can parse every string in S_{i-1}.

Suppose that w is w_i in $S_i (= S_{i-1} \cup \{w_i\})$. Then, from the manner of constructing UPDATE, since w is the first and only element of Q, the Claim implies that T_i can parse w. Hence, we have only to show that T_i can parse every string in S_{i-1}. But, to argue this, from the inductive hypothesis above, it suffices to show that T_i can parse every string in T_{i-1}. This is clearly true for each element of $T_{i-1} \cap T_i$. Now, consider the case (c) that can only replace $t_a^{(i-1)}$ in T_{i-1} with some prefix $t_a^{(i)}$ in T_i, where $t_a^{(i-1)} = t_a^{(i)} u$ and u is put at the end of Q to be later parsed. Then, by the Claim, u can be parsed by T_i, and since $t_a^{(i-1)}$ can be written as the concatenation of $t_a^{(i)}$ and u, we eventually conclude that every $t_a^{(i-1)}$ in T_{i-1} can be parsed by T_i. This completes the proof of the property (1).

(2) REFINE and PARSE

Let $T_i = \text{UPDATE}(T_{i-1}, w_i)$ and $M_{i-1} = (Q_{i-1}, T_{i-1}, \delta_{i-1}, p_0, F_{i-1})$ be the $(i-1)$-st conjecture. We describe how an SDA M_i is constructed from M_{i-1} and T_i. For that purpose, we need two subprocedures.

For a transition $t = (p, x, q) \in \delta_{i-1}$, we define **REFINE**$(T_i, M_{i-1}, t)$ by the set of transitions

$$\{(q_0, u_1, q_1), (q_1, u_2, q_2), \ldots, \; (q_{n-2}, u_{n-1}, q_{n-1}), (q_{n-1}, u_n, q_n)\},$$

where $q_0 = p$, $q_n = q$, $x = u_1 \cdots u_n$, and for each $j (1 \leq j \leq n)$ u_j equals t_a for some $t_a \in T_i$, and for each $j (1 \leq j \leq n-1)$ q_j is newly introduced if a transition (q_{j-1}, u_j, q_j) is not in δ_{i-1}.

Further, define **PARSE**(T_i, M_{i-1}, w_i) by the set of transitions

$$\{(q_0, u_1, q_1), (q_1, u_2, q_2), \ldots, (q_{n-2}, u_{n-1}, q_{n-1}), (q_{n-1}, u_n, q_n)\},$$

where $w_i = u_1 \cdots u_n$, and for each $j (1 \leq j \leq n)$ u_j equals t_a for some $t_a \in T_i$, and for each $j (1 \leq j \leq n)$ q_j is newly introduced if a transition (q_{j-1}, u_j, q_j) is not in δ_{i-1}, and $q_0 = p_0$ and q_n is marked as a new *accepting state* if it is not in F_{i-1}. (That is, this entails the construction of F_i.)

(3) CONSTRUCT

Let Δ_i be the set of all transitions defined by

$$\Delta_i = \bigcup_{t \in \delta_{i-1}} \text{REFINE}(T_i, M_{i-1}, t) \cup \text{PARSE}(T_i, M_{i-1}, w_i).$$

The procedure CONSTRUCT(Δ_i) takes as input Δ_i and returns a conjectured SDA M_i by merging equivalent states and edges in Δ_i using Lemma 2.

```
Procedure CONSTRUCT($\Delta_i$) :
    let $\Delta_i = \bigcup_{t \in \delta_{i-1}}$ REFINE($T_i, M_{i-1}, t$) $\cup$ PARSE($T_i, M_{i-1}, w_i$) ;
    merge identical states and edges in $\Delta_i$ using Lemma 2 ;
    let $\delta_i$ be the result of merging states and edges from $\Delta_i$ ;
    return $M_i = (Q_i, T_i, \delta_i, p_0, F_i)$, where $Q_i, F_i$ are obtained
        from $\delta_i$ in the obvious manner.
```

The complete algorithm LA is given below.

3.2. *Correctness of LA*

In this section we will prove the correctness of the presented algorithm LA. To this end, we start by defining an operation which is useful for our purpose.

Define a binary operation $\mathcal{O}$ on strings as follows : for $x, y \in \Sigma^*$,

$$\mathcal{O}(x,y) = \{u, v, w\}, \text{ where } u = \text{lcp}(x,y), x = uv, y = uw.$$

(Recall that lcp(x, y) denotes the longest common prefix of x and y.) Further, for a finite set of strings S, extend $\mathcal{O}$ by

$$\mathcal{O}(S) = \bigcup_{(x,y) \in S \times S} \mathcal{O}(x,y).$$

Then, consider the closure of S under $\mathcal{O}$ and denote it by $\mathcal{O}^*(S)$, i.e., let $\mathcal{O}^*(S) = \bigcup_{n \geq 0} \mathcal{O}^n(S)$, where $\mathcal{O}^0(S) = S$ and $\mathcal{O}^{n+1}(S) = \mathcal{O}(\mathcal{O}^n(S))$ (for each $n \geq 0$).

Let F be a finite set of strings over Σ^* where every symbol of Σ appears in some string in F. For each $a \in \Sigma$, F_a denotes the set of strings in F that start with a. Further, by min F we denote $\{sh(F_a) | a \in \Sigma\}$, where each $sh(F_a)$ is the set of shortest strings in F_a. (If $sh(F_a)$ is a singleton $\{f_a\}$, then we simply write f_a rather than $\{f_a\}$.)

We now *claim* that for each $a \in \Sigma$, if $\mathcal{O}^*(S)_a$ is not empty, then there is a unique shortest string x_a in it, i.e., $sh(\mathcal{O}^*(S)_a) = x_a$. Suppose there are shortest strings x_1 and x_2 in $\mathcal{O}^*(S)_a$ such that $x_1 = u_a v$, $x_2 = u_a w$, $len(v) = len(w)$ and $u_a = \text{lcp}(x_1, x_2)$. If $v \neq \lambda$, then $len(u_a) < len(x_1)$ and $u_a \in \mathcal{O}^*(S)_a$, contradicting the shortestness of x_1 in $\mathcal{O}^*(S)_a$. Hence, it must hold that $v = \lambda (= w)$, and we have that $x_1 = x_2$, thus proving the claim.

[Learning algorithm LA]

Input: a positive presentation of a strictly regular language L.

Output: a sequence of SDAs for strictly regular languages.

```
Procedure
    initialize $i = 0$ ;
    let $T_0 = \emptyset$ ;
    let $M_0 = (\{p_0\}, \emptyset, \emptyset, p_0, \emptyset)$ be the initial SDA ;
    repeat (forever)
      begin
        let $i := i + 1$ ;
        let $M_{i-1} = (Q_{i-1}, T_{i-1}, \delta_{i-1}, p_0, F_{i-1})$ be the current conjecture ;
        read the next positive example $w_i$ ;
        if $w_i \in L(M_{i-1})$ then output $M_i = M_{i-1}$ as the $i$-th conjecture;
        else
          begin
            let $T_i$ = UPDATE$(T_{i-1}, w_i)$ ;
            for all $t \in \delta_{i-1}$, make REFINE$(T_i, M_{i-1}, t)$ ;
                        /* refine old transitions using new edge labels */
            for $w_i$, make PARSE$(T_i, M_{i-1}, w_i)$ ;
                        /* parse $w_i$ using new edge labels */
            let $\Delta_i$ be the unions of each REFINE$(T_i, M_{i-1}, t)$  $(t \in \delta_{i-1})$
                and PARSE$(T_i, M_{i-1}, w_i)$ ;
            let $M_i$ =CONSTRUCT$(\Delta_i)$ ;
            output $M_i$ as the $i$-th conjecture ;
          end
      end
```

For each $i \geq 1$, let $S_i = \{w_1, ..., w_i\}$ be a sample set of a target SRL L. Let $T_{S_i} = \min \mathcal{O}^*(S_i) = \{x_a \mid a \in \Sigma\}$, where for each $a \in \Sigma$, $x_a = sh(\mathcal{O}^*(S_i)_a)$. T_{S_i} clearly has the strict prefix property. Also, from the strict prefix property of T_{S_i}, it is easy to show that any string w in S_i can be parsed by T_{S_i}, i.e., w can be written as the concatenation of elements of T_{S_i} with the property that $w = x_1 \cdots x_m = x'_1 \cdots x'_n$ $(x_j, x'_k \in T_{S_i}; 1 \leq \forall j \leq m, 1 \leq \forall k \leq n)$ implies that $m = n$ and $x_j = x'_j$ $(1 \leq \forall j \leq m)$.

Using this formalization, $T_i (= \text{UPDATE}(T_{i-1}, w_i))$ is characterized by T_{S_i} as follows

LEMMA 6 *Let $S_i = \{w_1, ..., w_i\}$. Then, for each $i \geq 1$, T_{S_i} coincides with T_i.*

Proof: By induction on i. Suppose $i = 1$, then $S_1 = \{w_1\}$ and $T_0 = \emptyset$. Since $\mathcal{O}(S_1) = \{\lambda, w_1\} = \{\lambda\} \cup S_1$, we have that $\mathcal{O}^*(S_1) = \{\lambda\} \cup S_1$. Hence, $T_{S_1} = \{x_a\} = T_1 (= \text{UPDATE}(T_0, w_1))$, where $x_a = w_1$ and a is the first symbol of w_1.

Since $S_{i-1} \subseteq \mathcal{O}^*(S_{i-1})$, it holds that $\mathcal{O}^*(S_{i-1} \cup \{w_i\}) \subseteq \mathcal{O}^*(\mathcal{O}^*(S_{i-1}) \cup \{w_i\})$. Conversely, since $\mathcal{O}^*(S_{i-1}) \cup \{w_i\} \subseteq \mathcal{O}^*(S_i)$, we have that $\mathcal{O}^*(\mathcal{O}^*(S_{i-1}) \cup \{w_i\}) \subseteq \mathcal{O}^*(\mathcal{O}^*(S_i)) = \mathcal{O}^*(S_i) = \mathcal{O}^*(S_{i-1} \cup \{w_i\})$. Thus, it is proved that

$$\min \mathcal{O}^*(S_{i-1} \cup \{w_i\}) = \min \mathcal{O}^*(\mathcal{O}^*(S_{i-1}) \cup \{w_i\}). \quad (1)$$

Suppose now that $T_{S_{i-1}} = T_{i-1}$, in other words, suppose that

$$T_{i-1} = \min \mathcal{O}^*(S_{i-1}) \quad \cdots \quad \text{(inductive hypothesis).}$$

By construction, since every string in $\mathcal{O}^*(S_{i-1})$ can be parsed by $T_{i-1} (= \min \mathcal{O}^*(S_{i-1}))$, it holds that

$$\min \mathcal{O}^*(\mathcal{O}^*(S_{i-1}) \cup \{w_i\}) = \min \mathcal{O}^*(\min \mathcal{O}^*(S_{i-1}) \cup \{w_i\}). \quad (2)$$

Then, noting that $T_i = \min \mathcal{O}^*(T_{i-1} \cup \{w_i\})$, we have

$$\begin{aligned} T_{S_i} &= \min \ \mathcal{O}^*(S_i) \\ &= \min \ \mathcal{O}^*(S_{i-1} \cup \{w_i\}) \\ &= \min \ \mathcal{O}^*(\mathcal{O}^*(S_{i-1}) \cup \{w_i\}) && \text{(by (1))} \\ &= \min \ \mathcal{O}^*(\min \mathcal{O}^*(S_{i-1}) \cup \{w_i\}) && \text{(by (2))} \\ &= \min \ \mathcal{O}^*(T_{i-1} \cup \{w_i\}) && \text{(by inductive hypothesis)} \\ &= T_i. \end{aligned}$$

Thus, the lemma is proved. ■

We also show that T_i is the "coarsest" finite set in the following sense.

LEMMA 7 *Let $T_i = T_{S_i}$ and let T' be any finite set of strings over Σ with the strict prefix property and such that every string in S_i can be parsed by T'. Then, every element in T_i can be parsed by T'.*

Proof: Since the $\mathcal{O}(X)$ operator adds only shorter strings to a set X, there is some finite k such that $\mathcal{O}^*(S_i) = \mathcal{O}^k(S_i)$. Now we show by induction on n that T' can parse every string in $\mathcal{O}^n(S_i)$. Then, since $T_i = T_{S_i} = \min \mathcal{O}^*(S_i) = \min \mathcal{O}^k(S_i)$, this immediately implies that every element in T_i can be parsed by T'.

By the hypothesis of the lemma, T' parses every string in $S_i = \mathcal{O}^0(S_i)$. Assume inductively that it parses every string in $\mathcal{O}^n(S_i)$. Then, the only strings in $\mathcal{O}^{n+1}(S_i) - \mathcal{O}^n(S_i)$ are triples of strings u, v and w such that uv and uw are in $\mathcal{O}^n(S_i)$, and the first characters of v and w are different. Thus, for such strings, T' parses uv and uw by the inductive hypothesis. Since T' has the strict prefix property, it is now not difficult to show that T' parses each of u, v and w, thus completing the proof.

■

We are now in a position to prove the following lemma.

LEMMA 8 *For each $i \geq 1$, M_i accepts the smallest SRL containing S_i. That is, $S_i \subseteq L(M_i)$ and for each $i \geq 1$, if L' is an SRL containing S_i, then $L(M_i) \subseteq L'$.*

Proof: Let $M_i = (Q_i, T_i, \delta_i, p_0, F_i)$ be the i-th conjecture. Then, by construction, it holds that $S_i \subseteq L(M_i)$. Let $M' = (Q', T', \delta', p'_0, F')$ be an acceptor such that $L' = L(M')$ containing S_i. By Lemma 7, each element of T_i can be written as a concatenation of elements of T'. Consider any u that is a prefix of a string w in S_i and is such that u can be written as a concatenation of elements of T_i. Then, since M' accepts w and u is a prefix of w that can be written as a concatenation of elements of T_i, $\delta'(p'_0, u)$ is a state in Q'.

Now, define a mapping f from Q_i to Q' as follows: For each $p \in Q_i$, let u_p be a prefix of a string in S_i such that the number of transitions in $\delta_i(p_0, u_p) = p$ is minimum. Then, define $f(p) = \delta'(p'_0, u_p)$. (As seen above, this is well-defined.) We show the following : (1) $f(p_0) = p'_0$, (2) if p is in F_i, then $f(p)$ is in F', (3) for every $p \in Q_i$ and every label string t such that $\delta(p, t) = q$ for some q, $f(\delta_i(p, t)) = \delta'(f(p), t)$.

By letting $u_{p_0} = \lambda$ and from the definitions of f and δ', we have

$$f(p_0) = \delta'(p'_0, u_{p_0}) = \delta'(p'_0, \lambda) = p'_0.$$

Suppose that p is in F_i. Then, since u_p is a prefix of a string in S_i, there exists a string v such that u_p and $u_p v$ are in $L(M_i)$ and that $u_p v$ is in S_i, where $\delta_i(p, v) \in F_i$. Let w_p be a string in S_i such that $\delta_i(p_0, w_p) = p \in F_i$. From the manner F_i is constructed using S_i and the minimality of u_p, u_p is in $L(M_i)$ implies that $u_p = w_p$ and, therefore, u_p is in S_i. Since S_i is contained in L', we have that u_p is in L'. That is, $f(p) = \delta'(p'_0, u_p)$ is in F'.

To prove (3), since t is a label string in T_i and t can be written as a concatenation of elements of T', there exists a state $\delta'(f(p), t)$ in Q'. For $q \in Q_i$ such that $\delta_i(p, t) = q$, let $u_q = u_p t$. Then, from the manner of choosing u_p, u_q is a prefix of a string in S_i such that the number of transitions in $\delta_i(p_0, u_q) (= \delta_i(p_0, u_p t)) = q$ is minimum. Further, from the manner δ_i is constructed, there is a string in S_i whose prefix is u_q. Therefore, we have that $f(q) = \delta'(p'_0, u_q)$. Then,

$$\begin{aligned}
\delta'(f(p), t) &= \delta'(\delta'(p'_0, u_p), t) \\
&= \delta'(p'_0, u_p t) \\
&= \delta'(p'_0, u_q) \\
&= f(q) \\
&= f(\delta_i(p, t)).
\end{aligned}$$

Thus, (1), (2) and (3) above immediately imply that $L(M_i) \subseteq L'$. ■

As a corollary, we have

COROLLARY 9 *Let L be a target SRL. Then, for each $i \geq 1$, it holds that $L(M_{i-1}) \subseteq L(M_i) \subseteq L$.*

In order to prove the correctness of the algorithm LA, we need to introduce the notion of a characteristic sample for a target SRL L. A finite subset S of L is called a *characteristic sample* of L if and only if L is the smallest SRL containing S, i.e., for any SRL L', $S \subseteq L'$ implies that $L \subseteq L'$ (Angluin, 1982).

Corollary 9 together with the definition of a characteristic sample almost immediately implies the following.

LEMMA 10 *For any $i \geq 1$, if S_i is a characteristic sample of L, then $L = L(M_i)$. Further, for any S if S is a characteristic sample of L, then so is any S_i containing S.*

We will show that there effectively exists a characteristic sample S_L of each SRL L.

We say that M is in *normal form* iff (1) it has no useless states, (2) every accepting sink state has exactly one transition defined into it, and (3) for every non-accepting state which is not the initial state, there are at least two transitions defined from it, where sink state is a state from which no transitions are defined.

Using an example, we demonstrate that SDAs in this form actually provide "normal forms" for SDAs.

Example 3 Consider an SDA M given in Figure 3. M has no useless states. For an accepting sink state p_4 that violates (2), duplicate it and introduce new state p_4'. For non-accepting states p_1 and p_3 that violate (3) and are not the initial state, we "haul up" each chain of transitions until the condition (3) holds, and delete p_1 and p_3 and, finally, replace old transitions involved in them with new transitions (p_0, ab, p_2), (p_2, db, p_2) and (p_2, cf, p_4'). Thus, a normal form M' is obtained such that M' is equivalent to M. □

It should be clear that any SDA can be transformed into such a normal form without changing the language.

Let $M = (Q, \Sigma, \delta, p_0, F)$ be an SDA in normal form that accepts L. We now construct a finite set S_M as follows. If M accepts only one string w, then let $S_M = \{w\} = L(M)$. Otherwise, we proceed below.

For each $p \in Q$, define the string u_p such that $\delta(p_0, u_p) = p$ in the following manner:

1. $u_{p_0} = \lambda$,
2. for any state p that is non-accepting or is not a sink state, let u_p be any string such that $\delta(p_0, u_p) = p$,

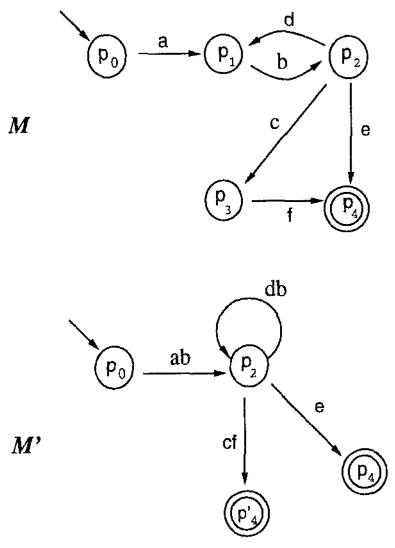

Figure 3. *M* and Its Normal Form *M'*

3. for each accepting sink state q such that $\delta(p,t) = q$, let $u_q = u_p t$, where t is the unique label of a transition from p into q.

Also, for each $p \in Q$, we choose any string v_p such that $\delta(p, v_p) \in F$. Finally, for each $q \in Q$, we define a set X_q consisting of one or two strings as follows:

(1) if q is an accepting sink state, then $X_q = \{\lambda\}$,

(2) if q is an accepting state with at least one transition defined from it, then choose one such transition (q,t,p) in δ, and $X_q = \{\lambda, tv_p\}$,

(3) if q is a non-accepting state (hence by the normal form, there are at least two transitions defined from it), then choose distinct transitions (q, t_1, q_1) and (q, t_2, q_2) in δ and let $X_q = \{t_1 v_{q_1}, t_2 v_{q_2}\}$,

(4) finally, if the initial state p_0 is non-accepting and has nothing but one transition defined from it, then $X_{p_0} = \{v_{p_0}\}$.

Now, we define a finite set S_M as follows :

$$S_M = \bigcup_{p \in Q} u_p X_p \cup \bigcup_{(p,t,q) \in \delta} u_p t X_q.$$

Example **4** Consider the SDA M' in Example 3. For each $p \in Q$, we choose u_p and v_p as follows:

$$\begin{array}{llll} u_{p_0} = \lambda, & u_{p_2} = ab, & u_{p_4} = abe, & u_{p'_4} = abcf \\ v_{p_0} = abe, & v_{p_2} = e, & v_{p_4} = \lambda, & v_{p'_4} = \lambda. \end{array}$$

(Note that we could instead choose $u_{p_2} = abdb,\ u_{p_4} = abdbe,\ u_{p'_4} = abdbcf$.)

Then, construct

$$X_{p_0} = \{abe\},\ X_{p_2} = \{e, cf\},\ X_{p_4} = \{\lambda\},\ X_{p'_4} = \{\lambda\}.$$

Finally, define $S_{M'}$ by

$$\begin{aligned} S_{M'} &= \{abe\} \cup \{abe, abcf\} \cup \{abe\} \cup \{abcf\} \\ &\quad \cup \{abe, abcf\} \cup \{abdbe, abdbcf\} \cup \{abe\} \cup \{abcf\} \\ &= \{abe, abcf, abdbe, abdbcf\}. \end{aligned}$$

□

LEMMA 11 *For any SDA M, S_M is a characteristic sample of $L(M)$.*

Proof: By construction, $S_M \subseteq L(M)$. We may assume that M accepts at least two strings. Let M' be the SDA M_{S_M} produced by LA when S_M is provided as a sample set. As we have seen in Lemma 8, since $L(M')$ is the smallest SDL containing S_M, we have that $L(M') \subseteq L(M)$. Suppose it is proved that $L(M) \subseteq L(M')$, then we have that $L(M) = L(M')$, which implies that $L(M)$ is the smallest SDL containing S_M. Therefore, it suffices to show that $L(M)$ is contained in $L(M')$.

Let $M = (Q, T, \delta, p_0, F)$ and $M' = (Q', T', \delta', p'_0, F')$ be SDAs described above. We show the following :

[Claim] For each $p \in Q$ and each t such that $(p, t, q) \in \delta$ for some $q \in Q$, both strings u_p and $u_p t$ lead to states of M' from p'_0.

(Proof.) If p is an accepting state, then u_p is clearly in S_M and, hence, in $L(M')$. Thus, u_p leads to a state of M'. If p is neither the initial nor an accepting state, then there are two strings $u_p t_1 v_{q_1}$ and $u_p t_2 v_{q_2}$ in $S_M (\subseteq L(M'))$, where t_1 and t_2 begin with different symbols. Since both are accepted by M', the longest common prefix of the two, u_p, leads to a state of M'. (Recall that if strings x and y are accepted by an SDA M, then the longest common prefix of x and y reaches a state of M from the initial state.) If p is the initial state p_0, then since $u_{p_0} = \lambda$, by definition, u_{p_0} trivially leads to a state (p'_0) of M'.

Further, suppose that p has a transition (p, t, q), then from the manner S_M is constructed, either there is a string or there are two strings of the form $u_p t v_q$ in $u_p t X_q (\subseteq S_M \subseteq L(M'))$. If q is an accepting state, then one can take $v_q = \lambda$ from X_q and

$u_pt(= u_ptv_q)$ is in $L(M')$. Thus, u_pt leads to a state of M'. If q is neither the initial nor an accepting state, then there are two strings v_1, v_2 in X_q such that they begin with different symbols. Since u_ptv_1 and u_ptv_2 are accepted by M', the longest common prefix u_pt of these two leads to a state of M'. Finally, if q is the initial state p_0 of M, then there is a non-empty string v_{p_0} in X_{p_0} such that $u_ptv_{p_0}$ is in $S_M(\subseteq L(M'))$. By definition, since v_{p_0} is in S_M, it is also in $L(M')$. From Corollary 3, this implies that u_pt leads to a state p'_0 of M' (End of the proof of Claim).

We now define a mapping f as follows : for each $p \in Q$, $f(p) = \delta'(p'_0, u_p)$. We would like to verify that (1) $f(p_0) = p'_0$, (2) if $p \in Q$ is an accepting state, then so is $f(p)$, and (3) for each transition (p, t, q), $f(\delta(p,t)) = \delta'(f(p), t)$. Since, by definition, $u_{p_0} = \lambda$, (1) obviously holds. If p is in F, then u_p is in S_M, and hence, in $L(M')$. Thus, $f(p)(= \delta'(p'_0, u_p))$ is in F'.

To show (3), it is sufficient to show that $f(q) = \delta'(f(p), t)$, since

$$\begin{aligned} f(\delta(p,t)) &= f(\delta(p_0, u_pt)) \\ &= f(\delta(p_0, u_q)) \\ &= f(q). \end{aligned}$$

To show that $f(q) = \delta'(f(p), t)$, we may equivalently show that $\delta'(p'_0, u_pt) = \delta'(p'_0, u_q)$.

Consider the cases for the state $q \in Q$: if q is an accepting sink state, then by definition, $u_q = u_pt$. Thus, the identity relation to be proved holds trivially. Otherwise, i.e., if q is a state with at least one transition defined from it, then X_q contains at least one non-empty string v' such that $\delta(q, v') \in F$. Then, there are two strings u_qv' and u_ptv' in $S_M(\subseteq L(M'))$. Since M' accepts these two strings, from Corollary 3, we have that $\delta'(p'_0, u_q) = \delta'(p'_0, u_pt)$, completing the proof that (3) holds.

From these properties (1), (2) and (3), we conclude that $L(M) \subseteq L(M')$ holds, which completes the proof of Lemma 11. ■

We are now in a position to prove the following theorem.

THEOREM 12 *The algorithm LA learns the class of SDAs in the limit from positive data.*

Proof: We prove that the algorithm LA converges to a correct SDA M accepting the target language L. We claim the following:
[Claim] For a characteristic sample S_L of L, if $S_L \subseteq S_{i_0}$ for some i_0, then $L = L(M_{i_0})$.
From Lemma 10, when the sample set S_{i_0} contains a characteristic sample S_L, S_{i_0} is also a characteristic sample of L. Then, from the property of a characteristic sample of L, it holds that $L \subseteq L(M_{i_0})$. On the other hand, by Corollary 9, we have that for each $i \geq 1$, $L(M_i) \subseteq L$, and hence, we conclude that $L = L(M_{i_0})$. Thus, the claim holds. Since S_L is finite, there exists $i_0 \geq 1$ such that $S_L \subseteq S_{i_0}$. From the claim, this completes the proof. ■

3.3. Time Analysis of LA

We analyze the time complexity of the algorithm LA.

[1] Time for Updating Conjectures

For each $i \geq 1$, let $M_i = (Q_i, T_i, \delta_i, p_0, F_i)$ be the SDA produced by LA and S_i be the set of positive examples at the i-th stage. Each time a new positive example w_i is given, updating the previous conjecture M_{i-1} is performed in time at most $O(\ell m)$, where $\ell = \mathrm{Max}_{w \in S_i}\{len(w)\}$, $m = |\Sigma|$. This is proved as follows :

(1) First, we consider UPDATE(T_{i-1}, w_i). An important observation about UPDATE is that only string matching operations are performed and nothing else is used throughout the procedure. Hence, all we have to do is to analyse how many lengths of strings, in total, must be scanned by UPDATE. Recall the construction of UPDATE. Each time case (c) occurs, some t_a in T is replaced with a proper suffix. (See Figure 4.) Thus, the number of times case (c) can occur is bounded by $x = \sum_{t_a \in T} len(t_a)$. Hence, the sum of the lengths of all of the strings ever put into Q (except for w_i) is at most x. Note that the length of each element of T is bounded by ℓ.

The algorithm halts after all strings ever put into Q (including the input string w_i) have been scanned. At each iteration of the **repeat** loop, at least one character of one of these strings is scanned as a prefix of the current string w in case (a), (b), or (c). Hence, the total number of iterations of the **repeat** loop is at most $len(w_i) + x$.

Thus, the total complexity of UPDATE(T_{i-1}, w_i) is bounded by:

$$\begin{aligned} & len(w_i) + \sum_{t_a \in T_{i-1}} len(t_a) \\ \leq\ & O(\ell m), \text{ where } \ell = \mathrm{Max}_{w \in S_i}\{len(w)\}. \end{aligned}$$

(2) First, for each $t = (p, u, q) \in \delta_{i-1}$ $(u \in T_{i-1})$, REFINE(T, M_{i-1}, t) is performed in time linear in $len(u)$. For PARSE(T, M_{i-1}, w_i), it requires at most linear time in $len(w_i)$. Thus, REFINE and PARSE require at most $\ell m + len(w_i)$.

In order to check if two transitions $(p, u, q), (p', u', q') \in \Delta_i$ are mergable or not, we have only to check the first symbols of u and u'. Since the cardinality of Δ_i is bounded by $\ell m + len(w_i)$, CONSTRUCT requires at most $\ell m + len(w_i)$.

Hence, we have the following :

THEOREM 13 *The algorithm LA may be implemented to run in time $O(|\Sigma|\ell)$, where ℓ is the maximum length of all positive data provided.*

[2] A Bound on the Number of Implicit Prediction Errors

We now consider the number of implicit errors of prediction needed for learning a correct SDA M accepting the target SRL L. First, we define the size of an SDA $M = (Q, T, \delta, p_0, F)$ based on the sum of the lengths of all elements in T, that is,

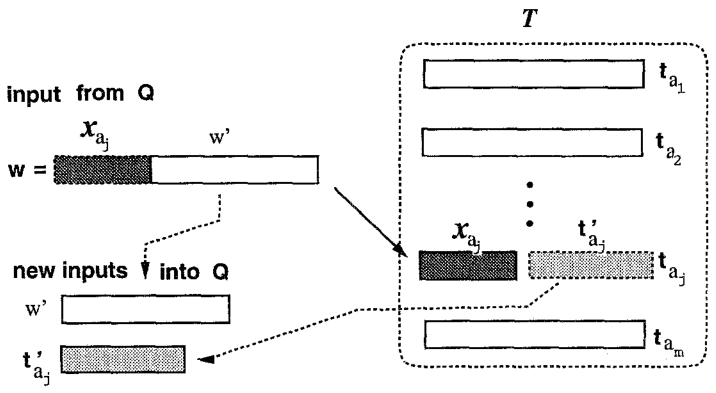

Figure 4. Updating T in UPDATE*(T,w)*

size$(M) = \sum_{u \in T}(len(u) + 1)$. This may be justified when we think of an equivalent minimum DFA in the usual sense.

LEMMA 14 *The number of implicit errors of prediction LA makes is at most $O(|\Sigma|\ell)$, where ℓ is the maximum length of all positive data provided.*

Proof: Let M be a correct SDA for the target L, and for each $i \geq 1$, let M_i be the i-th conjecture. Each time LA makes an implicit error of prediction, let us see what can happen to the conjecture in the algorithm LA. Suppose that $w (= w_i)$ is a positive example inconsistent with the previous conjecture M_{i-1}, i.e., $w_i \notin L(M_{i-1})$. First, since the set of transitions of each conjecture contains at most $|\Sigma|$ elements of the form (p, u, q), the maximum number of distinct states of any conjecture is bounded by $2|\Sigma|$. The following are possible cases where some changes to M_{i-1} must be made to produce a new conjecture M_i :

1. The set T is unchanged and exactly one non-accepting state of M_{i-1} becomes an accepting state of M_i. This can happen in total at most $2|\Sigma|$ times.

2. The set T is updated such that w causes at least one new element to be added to T. In this case, since the new elements added begin with characters that no other element of T began with, no states will be merged and exactly one new accepting state is added to M_{i-1}. Noting that the maximum number of occurrences of adding new elements to T is bounded by $|\Sigma|$, this can happen in total at most $|\Sigma|$ times.

3. The set T is refined such that at least one element of T is replaced by a proper prefix. Since the length of each element of T is bounded by ℓ, this can happen in total at most $|\Sigma|\ell$ times.

From these observations, we have that the number of implicit errors of prediction is bounded by $O(|\Sigma|\ell)$. ■

Noting that $|\Sigma| \leq \sum_{u\in T}(len(u)+1) = \text{size}(M) = n$, the following corollary follows.

COROLLARY 15 *The number of implicit errors of prediction LA makes is bounded by $O(\ell n)$, where n is the size of a correct SDA for the target language L.*

Thus, from Corollary 9, Theorems 12 and 13 and Corollary 15, we have the following theorem.

THEOREM 16 *The class of SDAs is polynomial-time learnable in the limit from positive data in the strict sense.*

Now, suppose that we have an algorithm $\mathcal{A}$ by which a class $\mathcal{M}$ is polynomial-time learnable in the limit from positive data in the strict sense (Definition 2). And, consider a learning model in which only equivalence queries are available for learning $\mathcal{M}$. From the learnability of $\mathcal{M}$ in the strict sense, we may assume that the algorithm $\mathcal{A}$ always confines its conjectures to subsets of the target language $L(M)$ for any M in $\mathcal{M}$. Then, equivalence queries can only return positive (counter)examples to $\mathcal{A}$. Further, the number of equivalence queries needed is bounded by that of implicit errors of prediction made by $\mathcal{A}$. Thus, the algorithm $\mathcal{A}$ can also achieve the polynomial-time learning of $\mathcal{M}$ using equivalence queries.

Under the same assumption, consider, in turn, a learning model in the PAC paradigm where equivalence queries can be replaced with a polynomial number of random sampling oracles $EX()$ (Angluin, 1987). Again, a polynomial number of implicit errors of prediction and the subset requirement of the algorithm $\mathcal{A}$ ensure that the number of random examples needed is bounded by a polynomial and the conjectures actually make no errors on negative data.

Thus, we have a corollary :

COROLLARY 17 *The class of SDAs is polynomial-time learnable using equivalence queries only. It is also PAC-learnable in polynomial time from positive examples only.*

3.4. Incremental Feature of LA

We have seen that the proposed algorithm LA is a polynomial-time algorithm that learns in the limit. Note that LA can actually learn a correct SDA M in polynomial time in the size of M and in the maximum length of data in the framework of learning in the limit. Although there are quite a few learnability results with polynomial-time efficiency, little is known about learning algorithms which achieve space efficiency as well. This is because it is generally hard to realize a learning strategy with restricted space, and most work proposed so far assumes unrestricted space for storing all the samples provided, in order to gain time efficiency. A learning algorithm is called *iteratively consistent* if and only if at each stage of learning, in order to make a new consistent guess it only uses one example and the current guess consistent with the data provided so far (Wiehagen, 1976; Jantke & Beick, 1981; Osherson, Stob, & Weinstein, 1986). Porat and Feldman (1991) discuss the learning problem of regular languages with finite storage. A learning algorithm *with finite working storage* is an algorithm that uses only finite working storage in addition to that for storing the current guess and the current example string. They showed that the class of DFAs is *not* learnable in the limit by any iteratively consistent algorithm with finite working storage. We note that the learning algorithm LA presented in the previous section has this nice property of iterative consistency with *finite* additional working storage. (In fact, LA can be implemented so that, besides storing the current guess and the current example, it requires at most $O(|\Sigma|)$ space for refining, parsing and merging operations.)

THEOREM 18 *The class of SDAs is polynomial-time learnable in the limit from positive data by an iterative, consistent algorithm with finite working storage.*

This is also in contrast to the negative result above for the whole class of DFAs.

4. Conclusions

In this paper, we have first proposed new definitions (Definitions 2 and 3) for polynomial-time learnability in the limit from positive data, and discussed the motivational background for introducing those concepts, by showing that any language class with an infinite descending chain property is not polynomial-time learnable in the limit from positive data in a simple definition (Definition 1) induced from Pitt's definition. Since many of the existing language classes have this property, this strongly suggested to "relax" the requirements of Definition 1 in some manner. In fact, Definition 2 has been defined as such a relaxation of Definition 1.

Then, after introducing a subclass of DFAs called strictly deterministic automata (SDAs), we have shown that the class of SDAs is iteratively, consistently polynomial-time learnable in the limit from positive data in the strict sense, that is, in the sense of Definition 3. Note that Definition 3 requires a stronger constraint called the subset requirement than Definition 2. As is well-known, the class of DFAs is neither learnable in the limit from positive data nor polynomial-time learnable in the limit in Pitt's definition (Pitt, 1989). Hence, the main result in this paper is in marked contrast to the fact above. (Note also that, besides iterative consistency and polynomial-time efficiency, the proposed learning algorithm has additional preferable features of conservativeness and responsiveness discussed in Angluin (1980).) As a corollary, it has been shown that the class of SDAs is polynomial-time learnable using equivalence queries only and it is also PAC-learnable in polynomial time.

Mäkinen (1990) discusses the problem of learning Szilard languages of linear grammars compatible with a given finite sample and gives a learning algorithm for solving the problem. It is interpreted that in the "in the limit" framework his algorithm runs in linear time for updating conjectures, but no discussion of implicit prediction errors is given. The class of Szilard languages of linear grammars is a proper subclass of the class of zero-reversible languages (Angluin, 1982). Note that the class of zero-reversible languages is not polynomial-time learnable using equivalence queries only (Angluin, 1990). On the other hand, although the class of strictly regular languages is incomparable to the class of zero-reversible languages, it properly contains the class of Szilard languages of linear grammars. (In fact, a Szilard language of a linear grammar is a language accepted by an SDA, with a single final state, having Σ as the set of labels.) Therefore, the main result in the present paper strengthens Mäkinen's result in two ways. As easily seen from the definition, the class of SDAs over an alphabet Σ strictly depends upon the size of Σ. However, it is the case that the class is infinite even if Σ is fixed. Further, the algorithm given in the present paper allows us to learn any SDAs with *arbitrary* alphabet size in time polynomial in the size of Σ, the size of a target SDA and the maximum length of all sample data. In other words, the algorithm works in polynomial time for the class of SDAs over a *growing alphabet.* This feature distinguishes our algorithm from most other existing ones. For future research, an interesting open problem is to find possible applications of the learnability results of SDAs to domains such as natural language acquisition (Berwick & Pilato, 1987), and syntactic pattern recognition in general (Gonzalez & Thomason, 1978).

Acknowledgments

The author would like to express his deepest gratitude to the referees for many useful suggestions and invaluable comments which significantly improved an early draft of the paper. In fact, many parts of this paper including Definition 2 and Theorem 1 have been contributed by the referees' efforts.

This work is supported in part by Grants-in-Aid for Scientific Research Nos. 03245104 and 04229105 from the Ministry of Education, Science and Culture, Japan.

References

Angluin, D. (1980). Inductive inference of formal languages from positive data. *Information and Control,* 45:117–135.

Angluin, D. (1982). Inference of reversible languages. *Journal of the ACM,* 29:741–765.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation,* 75:87–106.

Angluin, D. (1990). Negative results for equivalence queries. *Machine Learning,* 5:121–150.

Berwick, R.C. & Pilato, S. (1987). Learning syntax by automata induction. *Machine Learning,* 2:9–38.

Gold, E. M. (1967). Language identification in the limit. *Information and Control,* 10:447–474.

Gonzalez, R. C. & Thomason, M. G. (1978). *Syntactic Pattern Recognition: An Introduction.* Addison-Wesley, Reading, Mass.

Hopcroft, J. E. & Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts.

Jantke, K.P. & Beick, H-R. (1981). Combining postulates of naturalness in inductive inference. *Journal of Information Processing and Cybernetics,* 17:465–484.

Mäkinen, E. (1990). The grammatical inference problem for the Szilard languages of linear grammars. *Information Processing Letters,* 36:203–206.

Osherson, D.N., Stob, M., & Weinstein, S. (1986). *Systems That Learn: An introduction to learning theory for cognitive and computer scientists.* MIT Press, Cambridge, MA.

Porat, S. & Feldman, J.A. (1991). Learning automata from ordered examples. *Machine Learning,* 7:109–138, 1991.

Pitt, L. (1989). Inductive inference, DFAs, and computational complexity. In *Proceedings of 2st Workshop on Analogical and Inductive Inference,* Lecture Notes in Artificial Intelligence, Springer-Verlag 397, pp. 18–44.

Tanida, N. & Yokomori, T. (1992). Polynomial-time identification of strictly regular languages in the limit. *IEICE Transactions on Information and Systems,* E75-D:125–132.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM,* 27:1134–1142.

Wiehagen, R. (1976). Limeserkennung rekursiver funktionen durch spezielle strategien. *EIK,* pp.93–99.

Wood. D. (1987). *Theory of Computation.* Harper and Row, New York.

Received July 16, 1992
Accepted August 2, 1994
Final Manuscript August 22, 1994