Learning Local Languages and Their Application to DNA Sequence Analysis

Takashi Yokomori, *Member, IEEE*, and Satoshi Kobayashi

Abstract—This paper concerns an efficient algorithm for learning in the limit a special type of regular languages called strictly locally testable languages from positive data, and its application to identifying the protein α-chain region in amino acid sequences. First, we present a linear time algorithm that, given a strictly locally testable language, learns (identifies) its deterministic finite state automaton in the limit from only positive data. This provides us with a practical and efficient method for learning a specific concept domain of sequence analysis. We then describe several experimental results using the learning algorithm developed above. Following a theoretical observation which strongly suggests that a certain type of amino acid sequences can be expressed by a locally testable language, we apply the learning algorithm to identifying the protein α-chain region in amino acid sequences for hemoglobin. Experimental scores show an overall success rate of 95 percent correct identification for positive data, and 96 percent for negative data.

Index Terms—Local languages, deterministic automata, hemoglobin α-chain, DNA sequence analysis, machine learning.

1 Introduction

Grammatical inference can be vaguely described as the process of extracting from large quantities of data a grammatical representation for explaining those data. This is known as a subproblem of more general problem called inductive learning (or inductive inference) where instead of grammatical representations any type of concept representation devices is employed. One typical protocol of this learning paradigm, called *identification* in the limit, requires the class of objects (languages), the presentation of training data (an infinite sequence of examples) of an object in the class. The goal is to design an algorithm which produces a sequence of representations (grammars) for the object, eventually converging to a fixed representation correct for the targeted object. (See [4], [10] for excellent surveys of this area.)

In the study of inductive inference of formal languages, Gold [15] showed that the class of languages containing all finite sets and at least one infinite set is not learnable in the limit from positive data only. This fact was shocking in a sense because a simple implication is that even the class of regular languages is not learnable in the limit from positive data. Angluin [2] has given several conditions for the 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 $k \geq 0$ and shown these classes are learnable in the limit from positive data with the conjectures updated in polynomial time [3]. Here, the learnability from only positive data as well as polynomial-time efficiency is of great importance from the practical viewpoint.

On the other hand, the notion of a splicing system was introduced by Head in [18] as a mathematical model of restriction enzyme digestion and subsequent religation in the recombination of DNA molecules. He showed an interesting relationship between splicing languages generated by splicing systems and regular languages. In particular, one of the most significant results for our purpose is the equivalence relation between a certain type of splicing languages, called *persistent* splicing languages, and a subclass of regular languages called *strictly locally testable* languages. This result is crucially important in that, as we will show, it can bridge the gap between mathematical analysis in molecular biology and formal language theory in computer science. More specifically, in the usual problem setting of sequence analysis in genome informatics, the identification of protein sequence families is performed by the homology search technique to construct a "template" which gives a favorable score (e.g., [16], [32]). Our method is unique in that the learning algorithm in this paper may provide an automatic way of deriving such a template from sample data.

In this article, we will first show that, using Deterministic Finite State Automata (DFAs), for each $k \geq 1$, the class of strictly k-testable languages is learnable in the limit from positive data with the conjectures updated in linear time. It is also established that, for each $k \geq 1$, the class of strictly k-testable languages is a proper subclass of the class of $(k+1)$-reversible regular languages but incomparable to the class of zero-reversible languages. The class of strictly locally testable languages consists of all classes of strictly k-testable languages for every $k \geq 1$.

Then, motivated by a theoretical result due to Head [18] mentioned above which strongly suggests that a certain

- *T. Yokomori is with the Department of Mathematics, School of Education, Waseda University, 1-6-1 Nishi-waseda, Shinjuku-ku, Tokyo 169-8050, Japan. E-mail: yokomori@mn.waseda.ac.jp.*
- *S. Kobayashi is with the Department of Information Sciences, Faculty of Science and Engineering, Tokyo Denki University, Ishizaka, Hatoyama-cho, Hiki-gun, Saitama 350-0394, Japan. E-mail: satoshi@j.dendai.ac.jp.*

Manuscript received 12 Sept. 1994; revised 18 June 1998. Recommended for acceptance by S. Dunn.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 107053.

type of amino acid sequences can be expressed by a strictly locally testable language, we describe several experimental results via a machine identification system based on the learning algorithm developed above. We apply the system to identifying the protein α-chain region in amino acid sequences for hemoglobin. Some experimental data show that the system achieved an overall success rate of 95 percent correct identification for positive data, and 96 percent for negative data. As a result, we have specific DFAs which have been obtained through the learning process from the DNA sample data. Thus, the main experimental result stated in the present paper suggests a great potential of a successful formal method based on the automata-theoretic approach to DNA sequence analysis in general.

This article is organized as follows. After preparing basic definitions and notations in Section 2, we introduce the notion of a strictly k-testable language and study its formal properties in Section 3. A problem of learning strictly k-testable languages in the limit from positive data is formalized and a linear time algorithm for solving the problem is presented. Section 4 in turn deals with an application issue of the algorithm developed in the previous section. First, we state a theoretical result on relationship between two formal models: one for describing the behaviors of recombination of DNA molecules, called splicing system, and the other for a strictly k-testable language. Then, we apply the learning algorithm to an identification problem of protein α-chain region in amino acid sequences, to obtain high scores of overall success rates in several experiments. Finally, discussion on related works and concluding remarks follow in Section 5.

2 Preliminaries

2.1 Basic Definitions and Notations for Formal Languages

We assume the reader to be familiar with the rudiments of automata and formal language theory. (For notions and notations not stated here, see, e.g., [17].)

Let Σ be a fixed finite alphabet and Σ^* be the set of all finite-length strings over Σ. Further, let $\Sigma^+ = \Sigma^* - \{\lambda\}$, where λ is the null string. By $lg(u)$, we denote the length of string u. A *language over* Σ is a subset of Σ^*. The cardinality of a set S is denoted by $|S|$.

Let $L_k(w)$ and $R_k(w)$ be the prefix and the suffix of w of length k, respectively. Further, let $I_k(w)$ be the *set* of interior solid substrings of w of length k. These are defined only when w has length k or more. If w has length k, then $L_k(w) = R_k(w) = w$, while if w has length k or $k+1$, then $I_k(w)$ is empty. Let x be a string over Σ and L be a language over Σ. The *left-quotient* of L and x, denoted by $x\backslash L$, is defined by $x\backslash L = \{y \in \Sigma^* \mid xy \in L\}$.

A deterministic finite state automaton (DFA) is a five-tuple $M = (Q, \Sigma, \delta, p_0, F)$, where Q is a finite set of states, Σ is a finite alphabet of input symbols, p_0 ($\in Q$) is the initial state, $F(\subseteq Q)$ is the set of final (accepting) states, and δ is a state transition mapping such that for all $p \in Q$ and $a \in \Sigma$, $\delta(p, a)$ is in Q or undefined. A mapping δ is extended to $Q \times \Sigma^*$ as follows: For all $p \in Q$, $\delta(p, \lambda) = p$, and for all $p \in Q$, $x \in \Sigma^*$, and $a \in \Sigma$, $\delta(p, xa) = \delta(\delta(p, x), a)$. A string w over Σ is accepted by M if and only if (iff, for short) $\delta(p_0, w) \in F$. A language L is accepted by M, denoted by $L = L(M)$, iff L is exactly the set of all strings accepted by M.

2.2 Learning in the Limit From Positive Data

Let C be a subclass of regular languages to be learned over a fixed alphabet Σ and let $\mathcal{M}$ be a class of DFAs with the property that for each $L \in C$ there exists $M \in \mathcal{M}$ such that M accepts L (i.e., $L(M) = L$).

For a given $M \in \mathcal{M}$, *a positive presentation* of $L(M)$ is any infinite sequence of examples such that every $w \in L(M)$ occurs at least once in the sequence and no other examples not in $L(M)$ appear in the sequence.

Let L be a language in C such that $L = L(M)$ for some $M \in \mathcal{M}$. An algorithm $\mathcal{A}$ is said to *learn a language L in the limit from positive data* (using $\mathcal{M}$) iff for any positive presentation of L, the infinite sequence of DFAs M_i in $\mathcal{M}$ produced by $\mathcal{A}$ satisfies the property that there exists a *DFA* M' in $\mathcal{M}$ such that for all sufficiently large i, the ith conjecture (*DFA*) M_i is identical to M' and $L(M') = L(M)$. A class of languages C is *learnable in the limit from positive data* iff there exists an algorithm $\mathcal{A}$ that, given an L in C, learns L in the limit from positive data.

Let $\mathcal{A}$ be an algorithm for learning a language class C in the limit from positive data. A class C is *learnable in the limit from positive with the conjectures updated in polynomial time* iff there exists an algorithm $\mathcal{A}$ for learning C in the limit from positive data with the property that there exist a polynomial p such that for any n, for any L in C for which a correct DFA is of size n, and for any positive presentation of L, the time used by $\mathcal{A}$ between receiving the ith example w_i and outputting the ith conjectured representation r_i is at most $p(n, m_1 + \cdots + m_i)$, where $m_j = lg(w_j)(1 \le j \le i)$.

3 Learning Locally Testable Languages

In this section, we discuss in some detail the problem of learning a subclass of regular languages, called strictly locally testable languages, and present an algorithm for learning the class of languages in the limit from positive data. The learning algorithm is proven to be efficient enough to be applied to a practical problem domain, and in fact it is applied to an identification problem of amino acid sequences for hemoglobin in the next section.

3.1 Strictly Locally Testable Languages

Let k be a positive integer. A language L over Σ is strictly k-testable iff there exist finite sets A, B, and C such that A, B, $C \subseteq \Sigma^k$, and for all w with $lg(w) \ge k$, $w \in L$ iff $L_k(w) \in A$, $R_k(w) \in B$, and $I_k(w) \subseteq C$. (In this case, (A, B, C) is called a triple for L.) A language L is strictly locally testable iff there exists an integer $k \ge 1$ such that L is strictly k-testable [24]. Note that if L is strictly k-testable, then L is strictly k'-testable for all $k' > k$. Further, the definition of "strictly k-testable" says nothing about the strings of length $k-1$ or less.

For some analyses on the formal language theoretic characterizations of strictly locally testable languages, see the Appendix which establishes the relationships to other subclasses of regular languages.

Now, let us define a binary relation $\prec$ on $2^{\Sigma^k} \times 2^{\Sigma^k} \times 2^{\Sigma^k}$ as follows:

$(A, B, C) \prec (A', B', C')$ iff $A \subseteq A'$, $B \subseteq B'$, and $C \subseteq C'$.

It is easy to see that a relation $\prec$ is a partial order.

Let L be a strictly k-testable language over Σ and let (A, B, C) be a triple for L. A triple (A, B, C) for L is called *minimum* if (A, B, C) is the minimum element with respect to $\prec$ among all triples for L.

LEMMA 1. *Let L be a strictly k-testable language such that $L \subseteq \Sigma^k\Sigma^*$. Then, there exists a triple (A, B, C) for L which is minimum.*

PROOF. Suppose that there exist triples (A_i, B_i, C_i) for L which are pairwise incomparable with respect to the order $\prec$. Consider a triple (A, B, C), where $A = \cap_i A_i$, $B = \cap_i B_i$, $C = \cap_i C_i$. Then,

$$w \in L \Leftrightarrow (\forall i) L_k(w) \in A_i, R_k(w) \in B_i, I_k(w) \subseteq C_i$$
$$\Leftrightarrow L_k(w) \in \bigcap_i A_i, R_k(w) \in \bigcap_i B_i, I_k(w) \subseteq \bigcap_i C_i$$

Thus, (A, B, C) is a triple for L and clearly for all i, $(A, B, C) \prec (A_i, B_i, C_i)$. □

LEMMA 2. *Let L be a language such that $L \subseteq \Sigma^k\Sigma^*$. Then, L is strictly k-testable iff there exists a triple $[A',B',C']$ such that $A', B', C' \subseteq \Sigma^k$ and $L = A'\Sigma^* \cap \Sigma^*B' - \Sigma^+C'\Sigma^+$.*

PROOF. Suppose that there exists a triple (A, B, C) for L which is minimum. Then,

$$w \in L \Leftrightarrow L_k(w) \in A_i, R_k(w) \in B_i, I_k(w) \subseteq C$$
$$\Leftrightarrow w \in A'\Sigma^* \cap \Sigma^*B' \cap \left(\Sigma^* - \Sigma^+C'\Sigma^+\right)$$
$$\left(\text{where } A' = A, B' = B, \text{and } C' = \Sigma^k - C\right)$$

Thus, there exists a triple $[A', B', C']$ such that $A', B', C' \subseteq \Sigma^k$ and $L = A'\Sigma^* \cap \Sigma^*B' - \Sigma^+C'\Sigma^+$.

The converse is proved in the same manner as above. □

3.1.1 Conventions

1) First, recall that the definition of strictly k-testability says nothing about the strings of length $k-1$ or less. Since we are interested in the property on the essential part of strictly k-testable languages, we assume that all languages we will deal with in this section are subsets of $\Sigma^k\Sigma^*$. Further, note that a strictly k-testable language may possibly be strictly k'-testable for some k' less than k. In the present paper, we are concerned with the class of strictly k-testable languages where k is critical in the sense that no smaller k can specify the class. Hence, from Lemma 2, in what follows, we adopt the following definition for a strictly k-testable language: L is strictly k-testable iff
   a) there exists a triple $S = [A, B, C]$ such that $A, B, C \subseteq \Sigma^k$ and $L = L(S)$, and
   b) for no k' less than k, L is strictly k'-testable, where $L(S) = A\Sigma^* \cap \Sigma^*B - \Sigma^+C\Sigma^+$.
2) Second, a triple $[A, B, C]$ for L in this new definition) is called *minimum* iff so is a triple $(A, B, \Sigma^k - C)$ for L (in the original definition).
3) Third, we may assume that all triples $[A, B, C]$ considered and used in the proof are minimum.

3.2 Strictly *k*-Testable Language Associated With a Finite Set

In this subsection, we will show that given a finite set of strings T, we can effectively construct a strictly k-testable language L_T which is the smallest strictly k-testable language containing T.

Let T be a finite subset of $\Sigma^k\Sigma^*$. Then, construct finite sets A_T, B_T, and C_T as follows:

$$A_T = \{L_k(w) \mid w \in T\}$$

$$B_T = \{R_k(w) \mid w \in T\}$$

$$C_T = \{x \mid lg(x) = k, \forall y, y' \in \Sigma^+ (yxy' \notin T)\}.$$

Consider a triple $S_T = [A_T, B_T, C_T]$ and a language

$$L(S_T) = A_T\Sigma^* \cap \Sigma^*B_T - \Sigma^+C_T\Sigma^+.$$

Then, obviously, $A_T, B_T, C_T \subseteq \Sigma^k$. A set $L(S_T)$ is called a *strictly k-testable language associated with T*.

We assume that T and T' are finite subsets of $\Sigma^k\Sigma^*$ throughout this section.

LEMMA 3.

1) *$T \subseteq L(S_T)$ holds.*
2) *Let L be a strictly k-testable language such that $L \subseteq \Sigma^k\Sigma^*$ and $L = A\Sigma^* \cap B\Sigma^* - \Sigma^+C\Sigma^+$, for some triple $[A, B, C]$. Then, w is in L implies that $I_k(w) \cap C = \emptyset$.*
3) *If $T \subseteq T'$, then $L(S_T) \subseteq L(S_{T'})$ holds.*

PROOF. Statements 1 and 2 are obvious from the definition, and it suffices to prove Statement 3.

From the way of constructing A_T, B_T of S_T, it is obvious that $A_T \subseteq A_{T'}$, $B_T \subseteq B_{T'}$. Now, suppose that $C_{T'} \not\subseteq C_T$. Then,

$$\exists x \in C_{T'} - C_T \Rightarrow \forall y, y'(yxy' \notin T') \text{ and } \exists x, z'(zxz' \in T)$$
$$\Rightarrow \exists w(= zxz') \in T - T'$$
$$\Rightarrow T \not\subseteq T' \text{(contradiction)}$$

Thus, it must hold that $C_{T'} \subseteq C_T$. Hence, from the argument above, it is obvious that

$$A_T\Sigma^* \subseteq A_{T'}\Sigma^*$$

$$\Sigma^*B_T \subseteq \Sigma^*B_{T'}$$

$$\Sigma^* - \Sigma^+C_T\Sigma^+ \subseteq \Sigma^* - \Sigma^+C_{T'}\Sigma^+.$$

Hence, we have:

$$L(S_T) = A_T\Sigma^* \cap \Sigma^*B_T \cap (\Sigma^* - \Sigma^+C_T\Sigma^+)$$

$$\subseteq A_{T'}\Sigma^* \cap \Sigma^*B_{T'} \cap (\Sigma^* - \Sigma^+C_{T'}\Sigma^+)$$

$$= L(S_{T'}).$$

Thus, it holds that $L(S_T) \subseteq L(S_{T'})$. □

Note: If $T' \not\subseteq L(S_T)$, then $L(S_T) \subset L(S_{T'})$ *holds.*

LEMMA 4. *Let L be an arbitrary strictly k-testable language such that $L \subseteq \Sigma^k\Sigma^*$. Then, $T \subseteq L$ implies that $L(S_T) \subseteq L$.*

PROOF. Let $L = A\Sigma^* \cap \Sigma^*B - \Sigma^+C\Sigma^+$ for some triple $[A, B, C]$. Assume that $T \subseteq L$, then it is obvious that $(A_T \subseteq A)$ and $(B_T \subseteq B)$. Now, suppose that $C \not\subseteq C_T$. Then,

$$\exists \alpha \in C - C_T \Rightarrow \exists \alpha \in C, \exists y, y' \in \Sigma^+ (y\alpha y' \in T).$$

Since $T \subseteq L$, the string $y\alpha y'$ is in L. Thus, by (2) of Lemma 3, $I_k(y\alpha y') \cap C = \varnothing$. But, since $\alpha \in C$ and $\alpha \in I_k(y\alpha y') \cap C$, we have a contradiction. Thus, it holds that $C \subseteq C_T$. Hence, in the same manner as in the proof for (3) of Lemma 3, we have that $L(S_T) \subseteq L$. □

From Lemma 4, we note that $L(S_T)$ is the smallest strictly k-testable language containing T.

3.3 Characteristic Sample for Strictly k-Testable Language

We now consider a finite set R of a strictly k-testable language L with the property that L can be completely characterized by R.

Let L be a strictly k-testable language such that $L \subseteq \Sigma^k\Sigma^*$. A finite subset R of $\Sigma^k\Sigma^*$ is called a *characteristic sample for L* iff L is the smallest strictly k-testable language containing R.

LEMMA 5. *Suppose that $R \subseteq T \subseteq L$, where R and T are finite, L is a strictly k-testable language such that $L \subseteq \Sigma^k\Sigma^*$. If R is a characteristic sample for L, then $L = L(S_R) = L(S_T)$ holds.*

PROOF. From (1) of Lemma 3, it follows that $R \subseteq L(S_R)$. Hence, by the definition of characteristic sample for L, it must hold that $L \subseteq L(S_R)$. On the other hand, from Lemma 4, it holds that $L(S_R) \subseteq L$. Thus, we have

$$L = L(S_R). \tag{1}$$

Since $L(S_T)$ is the smallest strictly k-testable language containing T, $T \subseteq L$ implies that

$$L(S_T) \subseteq L. \tag{2}$$

Further, by (3) of Lemma 3, $R \subseteq T$ implies that

$$L(S_R) \subseteq L(S_T). \tag{3}$$

From (1), (2), and (3), we have that $L = L(S_R) = L(S_T)$ holds. □

Now, we will show that there effectively exists a characteristic sample for any strictly k-testable language.

3.3.1 Constructing a DFA Associated With a Triple

Let L be a strictly k-testable language such that $L = L(S)$ $(= A\Sigma^* \cap \Sigma^*B - \Sigma^+C\Sigma^+)$, for some triple $S = [A, B, C]$. We will show that there effectively exists a characteristic sample for L.

First, consider a *DFA* $M_S = (Q, \Sigma, \delta, p_0, F)$ constructed from $S = [A, B, C]$ in the following way:

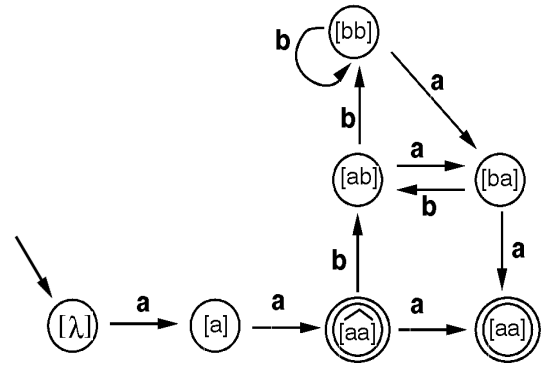


Fig. 1. Canonical DFA associated with S.

$$Q = \begin{cases} Q_1 \cup \{[\hat{x}] | x \in A \cap B \cap C\} & (\text{if } A \cap B \cap C \neq \varnothing) \\ Q_1 & (\text{otherwise}) \end{cases}$$

where

$$Q_1 = \{[\lambda], [a_1], [a_1a_2], ..., [a_1 \cdots a_{k-1}] \mid \alpha = a_1 \ldots a_k \in A\}$$

$$\cup \{[x] \mid x \in (\Sigma^k - C) \cup A \cup B\},$$

$$p_0 = [\lambda].$$

F and δ are defined as follows:

$$F = \begin{cases} \{[\beta] | \beta \in B\} \cup \{[\hat{x}] | x \in A \cap B \cap C\} & (\text{if } A \cap B \cap C \neq \varnothing) \\ \{[\beta] | \beta \in B\} & (\text{otherwise}) \end{cases}$$

(1) for $\forall \alpha = a_1 \ldots a_k \in A$,

$$\delta([p_0, a_1]) = [a_1]$$

$$\delta([w_i], a_{i+1}) = [w_i a_{i+1}] (\text{where } w_i = a_1 \ldots a_i (1 \leq \forall i \leq k-2))$$

$$\delta([w_{k-1}], a_k) = \begin{cases} [\hat{\alpha}] & (\text{if } \alpha \in B \cap C) \\ [\alpha] & (\text{otherwise}) \end{cases} (\text{where } w_{k-1} = a_1 \ldots a_{k-1})$$

(2) for $\forall [ax], [xb] \in Q$ s.t. $lg(x) = k-1, ax \in A, xb \in (\Sigma^k - C) \cup B$,

$$\delta([\widehat{ax}], b) = [xb] (\text{if } ax \in B \cap C)$$

$$\delta([ax], b) = [xb] (\text{otherwise})$$

(3) for $\forall [ax], [xb] \in Q$ s.t. $lg(x) = k-1$, $ax \in (\Sigma^k - C)$, $xb \in (\Sigma^k - C) \cup B$

$$\delta([ax], b) = [xb].$$

We call M_S the canonical DFA associated with S (see Fig. 1). It should be noted that the terminology *canonical* is used here in a manner different from the usual sense.

EXAMPLE 1. Consider a strictly two-testable language L defined by a triple $S = [A, B, C]$ as follows: $L = A\Sigma^* \cap \Sigma^*B - \Sigma^+C\Sigma^+$, where $\Sigma = \{a, b\}$, $A = B = C = \{aa\}$. Note that L is the language denoted by a regular expression: $aa + aa(b^+a)^*a$.

We construct the canonical DFA $M_S = (Q, \Sigma, p_0, \delta, F)$ associated with $S = [A, B, C]$ as follows:

$$Q = Q_1 \cup \left\{\left[\widehat{aa}\right]\right\}$$

where

$$Q_1 = \{[\lambda], [a], [aa], [ab], [ba], [bb]\}$$

$$p_0 = [\lambda]$$

$$F = \left\{\left[\widehat{aa}\right], [aa]\right\}.$$

δ is defined by

$$\delta([\lambda], a) = [a], \delta([a], a) = \left[\widehat{aa}\right], \delta\left(\left[\widehat{aa}\right], a\right) = [aa],$$

$$\delta\left(\left[\widehat{aa}\right], b\right) = [ab], \delta([ab], a) = [ba], \delta([ab], b) = [bb],$$

$$\delta([ba], a) = [aa], \delta([ba], b) = [ab], \delta([bb], a) = [ba],$$

$$\delta([bb], b) = [bb]$$

The state-transition graph of M_S is given in Fig. 1.

LEMMA 6. *M_S exactly accepts L, that is, it holds that for any $w \in \Sigma^k\Sigma^*$, w is in L iff it is accepted by M_S.*

PROOF. Let $lg(w) = k$, then from the definition of L and M_S,

$$w \in L \Leftrightarrow w \in A \cap B$$

$$\Leftrightarrow \begin{cases} \delta([\lambda], w) = [\hat{w}] \in F & (\text{if } w \in A \cap B \cap C) \\ \delta([\lambda], w) = [w] \in F & (\text{otherwise}) \end{cases}$$

Let $lg(w) = k + 1$ and $w = axb$(where $a, b \in \Sigma$, $lg(x) = k - 1$). Then,

$w \in L \Leftrightarrow ax \in A$ and $xb \in B$

$$\Leftrightarrow \delta([\lambda], ax) = \left[\widehat{ax}\right] \text{and } \delta\left(\left[\widehat{ax}\right], b\right) =$$

$$[xb] \in F(\text{if } ax \in B \cap C)$$

$$\left(\text{or}\,\delta([\lambda], ax) = [ax] \text{and } \delta([ax], b) = [xb] \in F\,(\text{otherwise})\right)$$

$$\Leftrightarrow \delta([\lambda], axb) = [xb] \in F$$

Now, assume that $lg(w) \geq k + 2$ and $w = a_1 \ldots a_n (n \geq k + 2)$. Then, $L_k(w) = a_1 \ldots a_k$, $R_k(w) = a_{n-k+1} \ldots a_n$, and $I_k(w) = \{in_2(w), \ldots, in_{n-k}(w)\}$, where $in_j(w) = a_j \ldots a_{j+k-1} (2 \leq \forall j \leq n - k)$. Then,

$$w \in L \Leftrightarrow L_k(w) \in A, R_k(w) \in B \text{ and } I_k(w) \subseteq \Sigma^k - C$$

$$\Leftrightarrow \delta([\lambda] = L_k(w)) = \left[L_k\widehat{(w)}\right], \delta\left(\left[L_k\widehat{(w)}\right], a_{k+1}\right) =$$

$$[in_2(w)](\text{if } L_k(w) \in B \cap C)$$

$$(\text{or}\,\delta([\lambda] = L_k(w)) = [L_k(w)], \delta([L_k(w)], a_{k+1}) =$$

$$[in_2(w)](\text{otherwise}))$$

$$\delta([in_j(w)], a_{j+k}) = [in_{j+1}(w)](2 \leq \forall j \leq n - k - 1) \text{ and}$$

$$\delta([in_{n-k}(w)], a_n) = [R_k(w)] \in F$$

$$\Leftrightarrow \delta([\lambda], w) = [R_k(w)] \in F$$

Thus, it is proved that for any $w \in \Sigma^k\Sigma^*$, $w \in L$ iff w is accepted by M_S. □

3.3.2 Constructing a Characteristic Sample

We observe that, for a given triple S for a strictly k-testable language L, the canonical DFA $M_S = (Q, \Sigma, \delta, p_0, F)$ associated with S is *not always* minimum.

Let $M_L = (Q_L, \Sigma, \delta_L, p_0, F_L)$ be the minimum DFA exactly accepting L, which is obtained from M_S by an appropriate state minimization algorithm.

For every state $q \in Q_L$, let pre(q) and post(q) be any strings of the minimum possible lengths such that $\delta_L(p_0, \text{pre}(q)) = q$ and $\delta_L(q, \text{post}(q)) \in F_L$, respectively. Further, for $q \in Q_L$ and a k-leader u of q, let $s(q, u)$ denote a string v of the minimum possible length such that $\delta_L(p_0, vu) = q$.

Let $Q'_L = \{[x] \in Q_L \mid lg(x) = k\}$. Then, consider a finite set:

$$R_{L,k} = \{\text{pre}([x])\text{post}([x]) \mid [x] \in Q_L - Q'_L\}$$

$$\cup \{s([x], x')x'\text{post}([x]) \mid [x] \in Q'_L, x' : k\text{–leader of } [x]\}$$

$$\cup \{s([x], x')x'b\text{post}([y]) \mid [x] \in Q'_L, \delta([x], b) = y, x' : k\text{–leader of } [x]\}$$

We remark that the manner of constructing $R_{L,k}$ from M_L faithfully follows that of constructing a characteristic sample for a k-reversible language L from the minimum DFA exactly accepting L [3].

LEMMA 7. *For any strictly k-testable language L, $R_{L,k+1}$ is a characteristic sample for L.*

PROOF. From Lemma 13 and the remark above, since L is $(k + 1)$-reversible, $R_{L,k+1}$ is a characteristic sample for L as a $(k + 1)$-reversible language. Let L' be any strictly k-testable language containing $R_{L,k+1}$. Since L' is a $(k + 1)$-reversible language, we have that $L \subseteq L'$. □

EXAMPLE 2. Consider the following strictly two-testable language L: $L = A\Sigma^* \cap \Sigma^*B - \Sigma^+C\Sigma^+$, where $\Sigma = \{a, b, c\}$, $A = \{ab\}$, $B = \{bc\}$, and $C = \{aa, bb, cc\}$. We construct the canonical DFA $M_S = (Q, \Sigma, p_0, \delta, F)$ associated with $S = [A, B, C]$ as follows:

$$Q = \{[\lambda], [a], [ab], [ac], [ba], [bc], [ca], [cb]\}$$

$$p_0 = [\lambda]$$

$$F = \{[bc]\}$$

$$\delta([\lambda], a) = [a]$$

$$\delta([a], b) = [ab]$$

$$\delta([xy], z) = [yz] \text{ (where } x \neq y, y \neq z, x, y, z \in \Sigma).$$

The state-transition graph of M_S is given in Fig. 2a. After minimizing M_S, we have the minimum DFA M_L depicted in Fig. 2b.

Note that since $A \cap B \cap C = \varnothing$, L is two-reversible. Actually, M_L in Fig. 2b satisfies the definition of k-reversible acceptor given in [3].

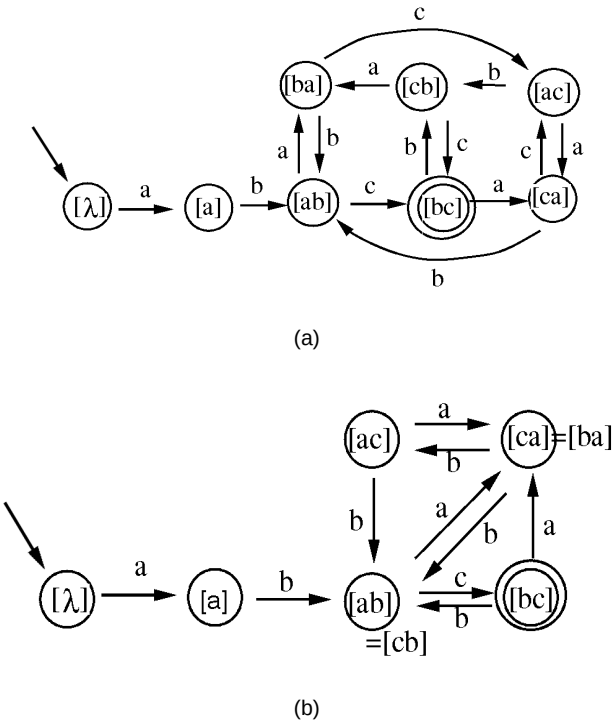


Fig. 2. (a) State-transition graph of M_S. (b) Minimum DFA M_L.

3.4 Learning Algorithm

We now present a learning algorithm (LA). The learning protocol for LA consists of only the positive presentation of an unknown strictly k-testable language U over the fixed alphabet Σ.

Suppose that a positive integer k and a positive presentation of a target language U are given. Then, the run of the algorithm LA is outlined as follows: After initializing necessary parameters, LA starts with producing the initial acceptor M_{E_0} accepting $E_0(=\varnothing)$. Each time of receiving a new positive example w_{i+1}, LA checks if the current conjectured DFA M_{E_i}, constructed from the current set of examples E_i, accepts w_{i+1} or not. If this is true, then LA simply produces M_{E_i} itself as the latest conjecture $M_{E_{i+1}}$. Otherwise, by examing w_{i+1} and computing each of $L_k(w_{i+1})$, $R_k(w_{i+1})$, and $I_k(w_{i+1})$, LA updates the current triple S_{E_i} so as to produce a new conjecture $M_{E_{i+1}}$ that is obtained from the updated triple $S_{E_{i+1}}$.

We show that LA given in Fig. 3 eventually learns in the limit a DFA M_E such that $U = L(M_E)$, where $L(M_E)$ is the language accepted by M_E.

LEMMA 8. *Let $M_{E_0}, M_{E_1}, \ldots M_{E_i}, \ldots$ be the sequence of DFAs produced by LA, where E_i is the set of positive data at the ith stage. Then,*

1) *for $\forall i \geq 0$, $L(M_{E_i}) \subseteq L(M_{E_{i+1}}) \subseteq U$, and*
2) *there exists $r \geq 0$ such that for $\forall i \geq 0$, $M_{E_r} = M_{E_{r+i}}$ and $L(M_{E_r}) = U$.*

PROOF. Since $E_i \subseteq E_{i+1}$, from (3) of Lemma 3, it holds that $L(S_{E_i}) \subseteq L(S_{E_{i+1}})$, hence $L(M_{E_i}) \subseteq L(M_{E_{i+1}})$. Further, by Lemma 4, we have that for $\forall i \geq 0$, $L(S_{E_i}) \subseteq U$, that is, $L(M_{E_i}) \subseteq U$.

Consider a sufficiently large integer $n_0 > 0$ such that $R_U \subseteq E_{n_0}$, where R_U is a characteristic sample for U. (Since R_U is a finite subset of U and from the definition of positive presentation, this is always possible.) Then, from Lemma 5, it follows that $U = L(S_{R_U}) = L(S_{E_{n_0}})$, hence $U = L(M_{R_U}) = L(M_{E_{n_0}})$. Further, once $M_{E_{n_0}}$ is output, since any input w given afterward is consistent with $M_{E_{n_0}}$, it is obvious that $(\forall i \geq n_0) M_{E_i} = M_{E_{n_0}}$. □

Thus, we have Theorem 9.

THEOREM 9. *Given an unknown strictly k-testable language U, the algorithm LA learns in the limit a DFA M_E such that $U = L(M_E)$.*

Input: a positive integer k and a positive presentation of a target strictly k-testable language U
Output: a sequence of DFAs for strictly k-testable languages

Procedure
initialize $E_0 = \varnothing$;
let $S_{E_0} = [\varnothing, \varnothing, \Sigma^k]$ be the initial triple;
construct DFA M_{E_0} accepting $E_0(=\varnothing)$;
repeat (forever)
  let $M_{E_i} = (Q_{E_i}, \Sigma, \delta_{E_i}, p_0, F_{E_i})$ be the current DFA;
  read the next positive example w_{i+1};
  if $w_{i+1} \in L(M_{E_i})$, **then** output $M_{E_{i+1}}$ $(= M_{E_i})$;
  else
    scan w_{i+1} to compute $L_k(w_{i+1}), R_k(w_{i+1}), I_k(w_{i+1})$;
    construct the canonical DFA $M_{E_{i+1}}$ associated with $S_{E_{i+1}} = [A_{E_{i+1}}, B_{E_{i+1}}, C_{E_{i+1}}]$;
    (where $A_{E_{i+1}} = A_{E_i} \cup \{L_k(w_{i+1})\}$,
    $B_{E_{i+1}} = B_{E_i} \cup \{R_k(w_{i+1})\}$,
    $C_{E_{i+1}} = C_{E_i} - I_k(w_{i+1})$
    and $E_{i+1} = E_i \cup \{w_{i+1}\}$)
  output $M_{E_{i+1}}$;

Fig. 3. Learning algorithm (*LA*).

3.5 Time Analysis of *LA*

Let $S_0 = [A_0, B_0, C_0]$ be the minimum triple for U and let $M_0 = (Q_0, \Sigma, \delta_0, p_0, F_0)$ be the canonical DFA associated with S_0. Further, let $M_U = (Q_U, \Sigma, \delta_U, p_0, F_U)$ be the minimum DFA obtained from M_0, where $U = L(S_0) = L(M_0) = L(M_U)$. (Note that given U, M_U is unique up to isomorphism.) We analyze the time complexity of the algorithm *LA*.

We may assume a $|\Sigma|$-branching complete tree T_Σ with depth k as a data structure for storing and handling k-length segments. Also, for every $i \geq 0$, the canonical DFA $M_{E_i} = \left(Q_{E_i}, \Sigma, \delta_{E_i}, p_0, F_{E_i}\right)$ obtained from a triple S_{E_i} is represented by a set of pairs $dom\left(\delta_{E_i}\right)\left(\subseteq Q_0 \times \Sigma\right)$. (This is possible because of the incremental nature of the constructed canonical DFAs. To indicate final states, we associate a label "f" with a state $[x]$ like $[x]_f$.)

For every $i \geq 0$, each time a new positive example w_{i+1} is provided, updating a conjecture M_{E_i} requires at most $O(k|\Sigma| lg(w_{i+1}))$. This is achieved as follows:

1) In the first phase, testing whether or not $w_{i+1} \in L\left(M_{E_i}\right)$ is done in time $O(lg(w_{i+1}))$.
2) In the second phase, computing $L_k(w_{i+1})$, $R_k(w_{i+1})$, and $I_k(w_{i+1})$ is done in time $O(k\, lg(w_{i+1}))$ by operating on T_Σ. At the same time, $dom\left(\delta_{E_i}\right)$ is updated in time $O(k|\Sigma| lg(w_{i+1}))$.

In the actual implementation, however, no triple is constructed, but $dom\left(\delta_{E_i}\right)$ is directly incremented. (Note that the "true" updating $dom\left(\delta_{E_i}\right)$ occurs in the second phase only when the first phase proves w_{i+1} to be inconsistent with the current conjecture. Further, because of the monotonically incremental feature of the output sequence of DFAs, the set $dom\left(\delta_{E_i}\right)$ increases itself just by adding either some elements of transitions or final states in a monotonical fashion.) Thus, we have Theorem 10.

THEOREM 10. *The algorithm LA requires at most $O(|\Sigma| km)$ time for updating a conjecture, where m is the maximum length among all positive data provided in the learning process.*

Note: If we are required to learn the minimum DFA, then it may take at most $\left|Q_{E_i}\right| \log \left|Q_{E_i}\right|$ additional time per update, using an efficient algorithm for this purpose [21], which is bounded by $|Q_0| \log |Q_0|$.

4 Applying *LA* to Biological Data—Experimental Results

In the first half of the present paper, we have developed an efficient algorithm *LA* for learning strictly locally testable languages in the limit from positive data and discussed some of the formal aspects of both the language class and the learning algorithm *LA*.

The second half of this section will discuss a possible application of such theoretical results to a practical problem in the domain of DNA sequence analysis. In the sequel, we first introduce a formal model for splicing DNA sequences and call one's attention to a theoretical fact that the language class characterized by a special type of the formal models is exactly the class of strictly locally testable languages. This immediately leads us to a kind of justification to apply our learning algorithm LA to the identification problem for biological data. In fact, we will present some experimental results which strongly suggest that this theoretical analysis is valid for a certain kind of amino acid sequences.

4.1 A Formal Model for DNA Splicing Sequences

In [18], Head proposed a formal system called a *splicing system* and studied the relationship between formal language theory and computational molecule biology.

A *splicing system* is a four-tuple $S = (\Sigma, I, B, C)$, where Σ is an finite alphabet, I is a finite set of initial strings, B and C are finite sets of triples (a, x, b), called *patterns*, with $a, x, b \in \Sigma^*$. For each such a triple, the string axb is called a *site* and the string x is called a *crossing*. Patterns in B and C are called the *left* (resp., *right*) *patterns*. For $uaxbv$ and $wcxdz$ in Σ^* with patterns (a, x, b) and (c, x, d) in the same hand (either B or C), two new strings $uaxdz$ and $wcxbv$ are constructed by splicing at the crossing x. (In a biological interpretation, one may take I as the initial set of DNA molecule sequences, B and C as the sets of splicing rules, specified by restricted enzymes and a ligase, that produce 5′ overhangs or blunt ends, and produce 3′ overhangs, respectively.)

The language $L = L(S)$ generated by S consists of the strings in I and all strings that can be obtained by adjoining to L $uaxdz$ and $wcxbv$, whenever $uaxbv$ and $wcxdz$ are in L and both patterns (a, x, b) and (c, x, d) are in the same hand. (In other words, $L(S)$ is the smallest subset of Σ^* which contains I and is closed under the operation of splicing.) A language L over Σ is a splicing language iff there exists a splicing system S such that $L = L(S)$.

Here we are concerned with a specific type of splicing systems. A splicing system is said to be *persistent* iff for each pairs of strings $uaxbv$ and $wcxdz$ in Σ^* with (a, x, b) and (c, x, d) from the same hand, if y is a subsegment of uax(resp., xdz) that is the crossing of a site in $uaxbv$(resp., $wcxdz$), then this same subsegment y of $uaxdz$ contains an occurrence of the crossing of a site in $uaxdz$. (Intuitively, in a persistent splicing system, it is possible to apply consecutive splicing operations infinitely many times.)

It is known that if we specify B and C by choosing restriction enzymes from actual biological world, then the resulting systems are often persistent [18]. (For example, in order to obtain persistent splicing systems, one may choose restriction enzymes from the [35, Appendix]. Further, any splicing system containing only one restriction enzyme is always persistent, even if the single enzyme is *Hgi***AI**.)

An important result for our purpose is Theorem 11.

THEOREM 11 [18]. *For a language L, the followings are equivalent*:

1) *L is a strictly locally testable language,*
2) *L is generated by a persistent splicing system.*

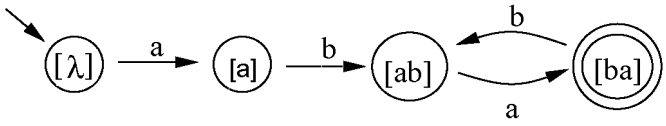


Fig. 4. A DFA accepting a strictly two-testable language denoted by a regular expression *aba*(*ba*)*.

EXAMPLE 3 [18].

1) Consider a splicing system $S = (\{a, b\}, \{ababa\}, \{(a, b, a)\}, \varnothing)$. It is easy to see that S is persistent and $L(S) = \{aba(ba)^n \mid n \geq 0\}$. Note that a string *aba* is obtained from *ababa* and its copy by using a pattern (a, b, a). A biochemical interpretation of this system is that taking $a = [A/T]\,[T/A]$ and $b = [C/G]\,[G/C]$, the resulting language $L(S)$ consists of all molecules that can be potentially arisen from the action of *Cla***I** (with an appropriate ligase) on (copies of) the molecule:

$$\left[\frac{A}{T}\right]\left[\frac{T}{A}\right]\left[\frac{C}{G}\right]\left[\frac{G}{C}\right]\left[\frac{A}{T}\right]\left[\frac{T}{A}\right]\left[\frac{C}{G}\right]\left[\frac{G}{C}\right]\left[\frac{A}{T}\right]\left[\frac{T}{A}\right] (= ababa).$$

A language $L(S)$ is, in fact, a strictly two-testable language that is accepted by a DFA pictured in Fig. 4.

2) Let $S = (\{\mathsf{a}, \mathsf{c}, \mathsf{g}, \mathsf{t}\}, I, B, C)$, where $\mathsf{a} = [A/T]$, $\mathsf{t} = [T/A]$, $\mathsf{c} = [C/G]$, $\mathsf{g} = [G/C]$, I is unspecified here, B consists of the patterns specified by *Eco***RI**, *Taq***I**, and *Alu***I**, and C contains the single pattern by *Hha***I**. The patterns provided by these four enzymes are as follows:

(g, aatt, c) for *Eco***RI**, (t, cg, a) for *Taq***I**

(ag, λ, ct) for *Alu***I**, (g, cg, c) for *Hha***I**.

A careful examination proves that this system is persistent.

We call {a, c, g, t} *the standard four-letter alphabet* and denote it by S_4. Note that in order to have a strictly two-testable language $L(S)$, we make use of a symbolic replacement in (1) of Example 3 above.

Such a *translation* is frequently used in analyzing DNA sequences. In fact, we will describe several experimental results in the next section where a letter-to-letter translation (i.e., a coding) is used to reduce the size of the problem in question smaller, leading to a feasible solution.

4.2 Protein α-Chain Identification

From Theorem 11, it may be justified that we assume the following Working Assumption.

A Working Assumption: Some types of amino acid sequences are generated by persistent splicing systems, and, hence, they are characterized by strictly locally testable languages.

Under this assumption, in this section, we describe some experimental results that concern identifying a protein α-chain region. We have implemented the learning algorithm for strictly locally testable languages developed in the previous section on a workstation, and applied it to the problem of identifying the α-chain region in amino acid sequences for hemoglobin.

Hemoglobin, as one of the most familiar proteins, provides an example of a protein consisting of more than one type of polypeptide chain: A heme group is associated with two α-subunits (α-chains) and two β-subunits (β-chains), where each type of subunit comprising a different polypeptide chain is represented by its own gene. Historically, it is believed that all globin genes are derived from a single ancestral gene, so that by tracing the development of individual globin genes we may learn about the mechanisms involved in the evolution of gene families. Here, we targeted the amino acid sequences for α-chains of hemoglobin, hoping to discover some formal language theoretic characterization property common in all of the amino acid sequences of this particular region.

TABLE 1
TRANSLATION TABLES
DAYHOFF'S CODING [9]

Amino Acids	Properties	New Symbols
C	sulfur polymerization	a
S, T, P, A, G	small	b
N, D, E, Q	acid & amide	c
H, R, K	basic	d
M, I, L, V	hydrophobic	e
F, Y, W	aromaticity	f

(a)

BINARY CODING [25]

Amino Acids	Hydropathy Index	New Symbols
A, C, F, G, I, L, M, N, S, T, V, W, Y	High	0
D, E, H, K, P, Q, R	Low	1

(b)

1) Goal: The purpose of this experiment is to find a DFA that identifies the α-chain region in amino acid sequences for *hemoglobin*. (As mentioned above, one hemoglobin molecule comprises two α-chains and two β-chains, and the α-chain is a polypeptide normally comprising 141 amino acids.)
2) Method: The positive data of α-chain in amino acid sequences for hemoglobin have been drawn from PRF protein sequence database [26]. Let us denote the set of those *raw* positive data by POS_{raw} that is a finite set of strings over the alphabet of 20 symbols, A_{20}, each of which represents each amino acid residue.

We used two kinds of letter-to-letter translations: One is from A_{20} to the six-letter alphabet D_6 (= {*a*, *b*, *c*, *d*, *e*, *f*}) specified by Dayhoff's coding method, the other from A_{20} to a binary alphabet B (= {0, 1}) used in [5], shown in Table 1. (A sample of a raw positive data and its translation results via Dayhoff's coding and binary coding are given in Fig. 5.)

We now describe the details of our experiments. Let's denote by POS the set of positive data obtained from POS_{raw} in terms of a translation via either Dayhoff's coding or binary coding in Table 1. In our experiments, the cardinality of POS was 123.

We have also prepared a set of *negative data* NEG, consisting of 2,567 sequences. NEG was constructed by randomly

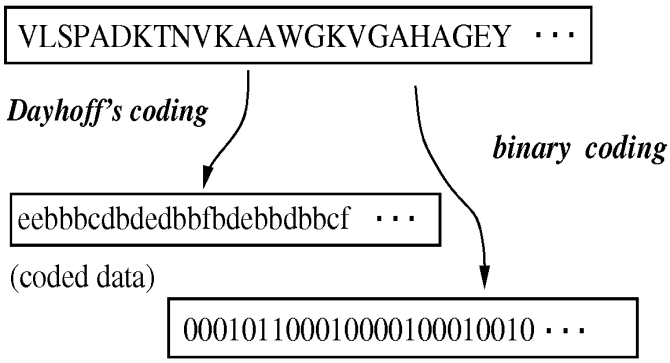


Fig. 5. Raw data of amino acid sequence from human hemoglobin is coded into new data over D_6 and B.

choosing from all protein data but hemoglobin data in PRF database. The length of data ranges from 138 to 142 for NEG and 141 or 142 for POS.

By **Exp**(k, x), we denote an experiment performed by the method described below, where k is the parameter for strictly k-testability and x is the cardinality of Sample, a set of *training* data *randomly* selected from POS.

4.2.1 Experiments Based on Dayhoff's Coding

We now have the new set of positive data POS over D_6, where a parameter k is fixed as $k = 2$.

An experiment Exp(2,x): For each x (= 10, 15, 20, 25, 30), we constructed Sample consisting of x sequences that were randomly selected from POS. Then, using Sample, the learning algorithm LA described in Section 3 produced a DFA $M_{2,x}$ that specifies a strictly two-testable language containing all training data in Sample. We then evaluated the performance of the obtained DFA, using the set of *test data* Test and NEG, where POS = Sample ∪ Test (disjoint union). The success rates of correct identification for positive and negative data are defined by:

$$\boldsymbol{R}_{pos} = \frac{\left|\text{Test} \cap L\left(M_{2,x}\right)\right|}{\left|\text{Test}\right|},\ \boldsymbol{R}_{neg} = \frac{\left|\text{NEG} \cap \overline{L\left(M_{2,x}\right)}\right|}{\left|\text{NEG}\right|}, \qquad (4)$$

respectively, where $\overline{L\left(M_{2,x}\right)}$ denotes the complement of $L(M_{2,x})$ with respect to D_6^* (that is, the set of strings over D_6 rejected by $M_{2,x}$).

For each x = 10, 15, 20, 25, 30), we have iteratively made an experiment Exp(2, x) 100 times. Fig. 6 illustrates a series of experiments Exp(2, x) where the figures indicate the *average* scores on each experiment. Table 2 summarizes the experimental results. Table 2a shows that the DFA $M_{2,30}$ achieved a success rate of 93 percent correct identification for (positive) test data and 99 percent for negative data.

4.2.2 Experiments Based on Binary Coding

Using a binary coding in [25], we have a set of positive data POS over B which is obtained from POS_{raw}.

By fixing x = 20, we first made experiments Exp(k, 20) for each k = 2, 3, 4, 5, and 6. In each case, Exp(k, 20) has been iteratively performed 100 times. The final results are presented in Table 2b, where R_{pos} and R_{neg} are defined in the same manner as in (4), and all figures indicate the average

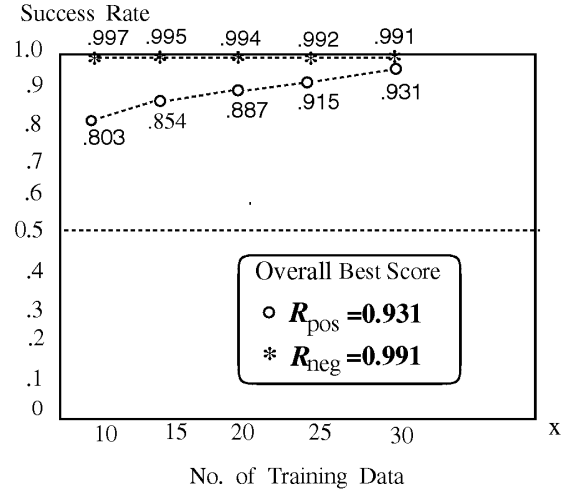


Fig. 6. In experiments Exp(2, x) over D_6, where x ranged from 10 to 30, the best score was attained in Exp(2, 30). The scores indicate the average rates of 100 random experiments.

scores. For example, we observe that an experiment Exp(3, 20) achieves the overall best score: a success rate of 96 percent correct identification for (positive) test data and 92 percent correct identification for negative data.

Based on this observation, we then made experiments Exp(3, x) for each x = 6, 7, 8, 9, and 10. Again, for each x, Exp(3, x) has been performed 100 times. Fig. 7 illustrates the results of a series of Exp(3, x), where the figures indicate the average scores.

4.3 Experimental Results

First, we will mention some theoretical consideration and its molecule biological implication that are immediately derived from our previous results.

Suppose that a representation relation $L = h(L')$ holds, where L is a strictly k-testable language over D_6 and h is a letter-to-letter coding from A_{20} to D_6. Then, we note that L' is contained in a k-testable language over A_{20}. Further, if a set of amino acid sequences is a strictly k-testable language over A_{20}, then it is regarded as 3k-testable language over the standard four-letter alphabet S_4 (= {a, c, g, t}).

From these considerations, Theorem 12 holds.

THEOREM 12. *If L is a strictly k-testable language over D_6, then a language L' such that $L = h(L')$ is contained in a 3k-testable language over S_4.*

TABLE 2
EXPERIMENTAL RESULTS—BEST SCORES VIA DAYHOFF CODING

(k = 2)	Exp(2, 30)
R_{pos}	93%
R_{neg}	99%

(a)

VIA BINARY CODING BASED ON HYDROPATHY INDEX

Exp(k, 20)	k = 2	k = 3	k = 4	k = 5	k = 6
R_{pos}	97.4%	96,2%	92.6%	87.9%	81.5%
R_{neg}	83.3%	92.3%	93.6%	94.9%	99.2%

(b)

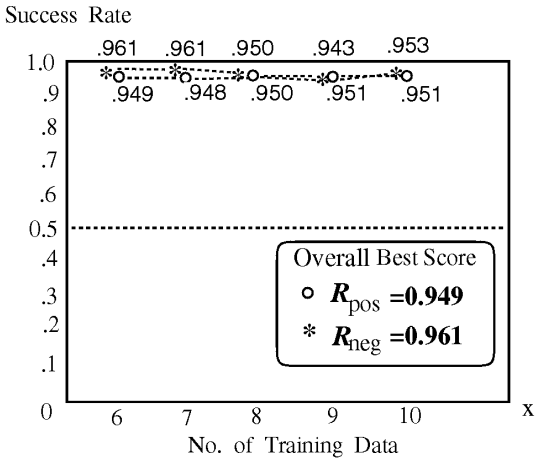


Fig. 7. In experiments Exp(3, *x*) over B, where *x* ranged from six to 10, the best score was attained in Exp(3, 6). The scores indicate the average rates of 100 random experiments.

Thus, for example, from a result obtained in Exp(2, 30) with $k = 2$, one may make a conjecture that *a set of DNA sequences for α-chain region of hemoglobin is contained in a strictly six-testable language over* Σ_4, which is strongly supported by the theoretical discussion made in Section 4.1.

4.4 α-Chain Automata

Through the experiments, several DFAs have been identified from sets of training data that have been selected in a random manner from a set of positive data POS. It might be expected that these DFAs have some common feature, because they all have been obtained as the best candidate DFA for specifying the same virtual target of a strictly locally testable language (i.e., a persistent splicing language).

Let us first focus on the series of DFAs over B, $M_{k,20}$ (k = 2, 3, 4, 5, 6), obtained in Exp(k, 20). Provided that the overall performance score of a DFA is evaluated by a simple summation, $R_{pos} + R_{neg}$, we observe that $M_{3,20}$ may be called the best DFA over B. We also note from Fig. 7 that all DFAs $M_{3,x}$ (for x = 6, 7, 8, 9, 10) equally achieve high scores in overall performance evaluation. This suggests one aspect of the stability of the obtained DFA family. Further, it is somehow surprising that only six (and at most 10) training data *randomly* selected are sufficient for constructing such a good DFA characterizing α-chain region in binary representation. More remarkable is that among 100 times of random experiments in Exp(3, 6), we have obtained more than 70 times the identical DFA $M_{3,6}$, accepting $000(0 + 1)^*101$, shown in Fig. 9. Thus, an interesting fact is that with high probability of 95 percent, the structure of α-chain region of amino acid sequence can be characterized by only its prefix and suffix of length three, in *binary* representation in terms of hydropathy index.

Similarly, we have observed from the series of experiments in Exp(2, x) over D_6 (for x = 10, 15, 20, 25, 30) that the best score was achieved in Exp(2, 30) (see Fig. 6), where the best DFA $M_{2,30}$ shown in Fig. 8 was obtained 10 times out of 100 random experiments. This DFA is characterized by the fact that each acceptable string starts with either *be* or *ee* and ends with *fd*, in the meanwhile, no *aa* appears as an internal substring.

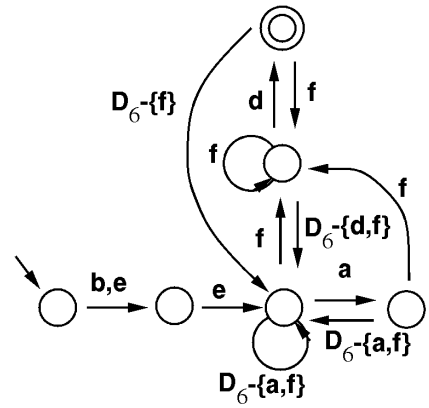


Fig. 8. A DFA $M_{2,30}$ most frequently obtained in experiments Exp(2, 30) over D_6. This DFA was obtained more than 70 among 100 times of random experiments.

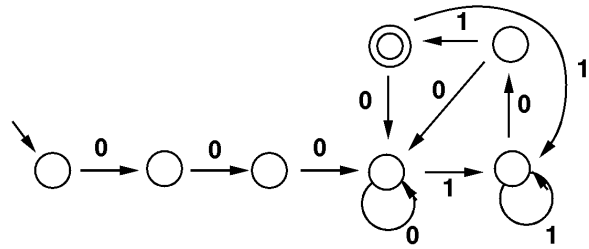


Fig. 9. A DFA $M_{3,6}$ most frequently obtained in experiments Exp(3, 6) over B. This DFA was obtained more than 10 among 100 times of random experiments.

One might think of comparing two DFAs $M_{3,6}$ over B and $M_{2,30}$ over D_6 obtained above. However, we find it difficult to discuss one's superiority over the other, because these experiments have been performed based on the distinct training data sets *randomly chosen* under the different coding methods. Further, two codings we used are basically independent of each other. Thus, it seems that, at present, we have no justifiable way to select one DFA as the best automaton for explaining α-chain in amino acid sequences.

In summing up, using the proposed method for learning strictly locally testable languages, we could obtain a specific type of DFA M_α that one may call an *α-chain DFA* for hemoglobin. A DFA M_α so obtained is applicable to identifying the α-chain region for unknown amino acid sequences with a very high success rate.

Finally, from the other supplemental experiments, we conjecture that a DFA with a similarly high score could be obtained from a set of positive *raw* data over Σ_{20}, i.e., POS_{raw} if the training set is sufficiently large. Here, we could eventually collect only 123 positive raw data in total from the PRF database, and, as a result, we could not help using two codings to obtain our main results.

5 Discussion

5.1 Related Works

In [3], Angluin presents an algorithm that, given $k \geq 1$, learns k-reversible DFAs in the limit from positive data

where a conjecture is updated in $O(kN^3)$ time, where N is the sum of lengths of data provided. For the relationship between strictly locally testable languages and reversible languages, see the Appendix. There are some works by Garcia and others [12], [11] devoted to the study of the identification problem of strictly k-testable languages, where the definition of strictly k-testability is slightly different from the one in this paper. In [11], a language L is strictly k-testable iff there exists a triple [A, B, C] such that

$$A, B \subseteq \bigcup_{i=1}^{k-1} \Sigma^i,$$

$$C \subseteq \Sigma^k,$$

and

$$L = A\Sigma^* \cap \Sigma^* B - \Sigma^* C \Sigma^*.$$

This definition of strictly k-testability leads to another class of strictly k-testable languages. (For example, a strictly one-testable, and therefore, two-testable language $L = \{ab^n a \mid n \geq 0\}$ in our definition is neither strictly one-testable nor two-testable in their definition.) They present an algorithm which is quite similar to the LA in this paper and runs in time $O(kN \log |Q_0|)$, where Q_0 is the state set of the canonical DFA associated with a triple for the target language. The work is, however, achieved from the viewpoint different from the one in this paper in that their efforts mainly put stress on the construction of the canonical DFA associated with a triple from a given sample set. They also apply their learning algorithm to the syntactic pattern (speech) recognition problem [11].

There exist a few subclasses of regular languages known to be learnable in the limit from positive data in *polynomial time* in some sense, such as the class of k-reversible languages, the class of regular pattern languages [29], the class of k-bounded regular languages [22]. One of the papers by Shinohara [30] also suggests another subclass of regular languages of special type [34], [36] polynomial-time learnable in the limit from positive data. It seems that further research in this direction should be made for enlarging the boundary of polynomial-time learnability from positive data.

Since the notion of a splicing system was introduced by Head in [18] as a mathematical model of restriction enzyme digestion and subsequent religation in the recombination of DNA molecules, there have been reported an enormous number of works concerning the formal characterizations of splicing systems and their learning problem [8], [13], [14], [19], [31], [33], [38]. Among others, one of the most significant results for our purpose is the equivalence relationship between persistent splicing languages and strictly locally testable languages established in [18]. This equivalence relation encouraged and provided us with a kind of justification to apply our learning algorithm to the identification problem of biological data.

In the area of machine learning (ML), there have been made many attempts to classify, identify and predict the biological properties of DNA and protein sequences. It should be remarked, however, that in the usual problem setting of DNA or protein sequence analysis in the ML community, the identification of protein sequence families is often performed by the homology search technique to construct a "template" which gives a favorable score (e.g., [16], [32]). Among others, there are some works motivated by the linguistic characterizations of DNA and protein sequence analysis: Searls [28] champions the use of formal languages and gives an intensive investigations on computational linguistics for biological sequences. Also, one can find a variety of proposals for the use of pattern languages (e.g., [20], [25]) and of probabilistic context-free grammars and/or its extensions (or variants) for modeling biological sequences (e.g., [1], [6], [27], [23], [37]). However, none of those works has attempted to deal with learning issues of the biological template. In this regard, our method is unique, that is, the learning algorithm in this paper may provide an automatic way of deriving such a template from sample data.

5.2 Concluding Remarks

We have shown that the class of strictly k-testable languages is learnable in the limit from positive data and presented an algorithm which learns any strictly k-testable language in the limit from positive data using DFAs.

It should be noted that similar arguments might work for the class of locally testable languages, which is a larger subclass of regular languages, and lead us to the similar results. As Angluin suggested [3], it is interesting to study on the relationship between the learnability and the algebraic properties of the subclasses of regular languages. The class of noncounting regular languages is identical to the smallest class of languages that contains all locally testable languages and closed under Boolean operations and concatenation [24]. Alternative characterizations for noncounting regular languages are due to the group-freeness and the permutation-freeness on their syntactic monoids. These algebraic properties also have some connections to a certain subclass of first-order logic.

Motivated by a result due to Head [18], which strongly suggests that a certain type of amino acid sequences can be expressed by a strictly locally testable language, we have described some experimental results via a machine identification system based on the learning algorithm developed above. We applied the system to the problem of identifying the protein α-chain in amino acid sequences for hemoglobin. The main experimental data showed that the system achieved an overall success rate of 95 percent correct identification for positive data and 96 percent for negative data. Under the same working assumption made here, the prediction problem of other domains such as protein *α-helix* region in amino acid sequences could be also attacked in terms of the same identification strategy.

Finally, as is mentioned above, strictly locally testable languages have some connections to a certain subclass of first-order logic formulas. In other words, they can provide a certain class of logical formulas for describing the genetic information. In this sense, the main experimental result stated in this article has a deep implication to the bio-informatics aspect of DNA sequences and also suggests a great potential of a successful formal method based on the automata-theoretic approach to DNA sequence analysis in general.

APPENDIX

In order to discuss the relationship between the class of strictly locally testable languages and that of reversible languages, we give the definition of reversible languages based on the language-theoretic characterizations [3]. Let k be a nonnegative integer. A regular language L is *k-reversible* iff whenever u_1vw and u_2vw are in L and $lg(v) = k$, it holds that $u_1v\backslash L = u_2v\backslash L$. (When $k = 0$, we write "zero-reversible" rather than "0-reversible.") A regular language is *reversible* iff it is k-reversible for some $k \geq 0$.

EXAMPLE 4. Consider a language L consisting of all finite-length strings over $\Sigma(= \{a, b\})$ that begin with a, end with a, and contain no occurrence of a as a strictly interior subword, that is, L is the language denoted by the regular expression $a + ab^*a$.

Let

$$A = B = \{a\}, \text{ and } C = \{b\}.$$

Then, it is easily seen that for $\forall w \in \Sigma^*$ with $lg(w) \geq 1$, $w \in L$ iff $L_1(w) \in A$, $R_1(w) \in B$, and $I_1(w) \subseteq C$. Hence, L is strictly one-testable. It is seen that L is not a zero-reversible language.

On the other hand, the language denoted by the regular expression $a(bb)^*$ is not strictly k-testable for any $k \geq 1$, but is zero-reversible. Further, there are some languages (e.g., $\{ab^n \mid n \geq 1\}$) which are strictly one-testable as well as zero-reversible. Hence, the class of strictly k-testable languages is incomparable to the class of zero-reversible languages. (There exists another definition for strictly k-testability [7], [12], [11] that is slightly different from the one given in the present paper. This has been discussed in the Section 5.1.)

In fact, we can show the following characterization of strictly k-testable languages which is similar to but different from Theorem 7.1 in [12], and the proof is also performed in a similar fashion.

LEMMA 13. *For each $k \geq 1$, a strictly k-testable language L is a $(k + 1)$-reversible language.*

PROOF. Let (A, B, C) be a triple for L over Σ. It suffices to show that whenever u_1vw and u_2vw are in L and $lg(v) = k + 1$, it holds that $u_1v\backslash L = u_2v\backslash L$.

Suppose that u_1vw, u_2vw are in L and $lg(v) = k + 1$. Then, we have that $L_k(u_iv) = L_k(u_ivw) \in A$, $R_k(vw) = R_k(u_ivw) \in B$, and $I_k(u_ivw) = I_k(u_iv) \cup I_k(vw) \subseteq C (i = 1, 2)$. In particular, it holds that for $i = 1, 2$, $I_k(u_iv) \subseteq C$. Then, for $\forall x \in \Sigma^*$,

$$\begin{aligned} I_k(u_1vx) \subseteq C &\Leftrightarrow I_k(u_1v) \cup I_k(vx) \subseteq C \\ &\Leftrightarrow I_k(u_2v) \cup I_k(vx) \subseteq C \\ &\Leftrightarrow I_k(u_2vx) \subseteq C \end{aligned}$$

Using these relations, we can show that for $\forall x \in \Sigma^*$, u_1vx is in L iff u_2vx is in L. Actually, for $\forall x \in \Sigma^*$,

$$\begin{aligned} u_1vx \in L &\Leftrightarrow L_k(u_1vx) \in A, R_k(u_1vx) \in B, I_k(u_1vx) \subseteq C \\ &\Leftrightarrow L_k(u_2vx) \in A, R_k(u_2vx) \in B, I_k(u_2vx) \subseteq C \\ &\Leftrightarrow u_2vx \in L \end{aligned}$$

Hence, we have that $u_1v\backslash L = u_2v\backslash L$. □

Note: For each $k \geq 1$, there exists a strictly k-testable language $L(k)$ that is not k-reversible. For example, let $A = B = \{a^k\}$ and $C = \Sigma^k - \{a^k\}$, where $\Sigma = \{a, b\}$, and let $L(k)$ be a language defined by a triple (A, B, C). Then, $\{a^k, a^{k+1}\} \subseteq L(k)$ and $\{a^{k+i} \mid i \geq 2\} \cap L(k) = \emptyset$. Let $u_1 = \lambda$, $u_2 = a$, $v = a^k$, $w = \lambda$, then both $u_1vw = a^k$ and $u_2vw = a^{k+1}$ are in $L(k)$. However, $u_1v\backslash L(k) \supseteq \{\lambda, a\}$, while $u_2v\backslash L(k) = \{\lambda\}$, thus, $u_1v\backslash L(k) \neq u_2v\backslash L(k)$. Hence, $L(k)$ is not k-reversible.

Thus, we have Theorem 14.

THEOREM 14. *For each $k \geq 1$, the class of strictly k-testable languages is properly included in the class of $(k + 1)$-reversible languages, but incomparable to the class of zero-reversible languages.*

We present another characterization of strictly k-testable languages which provides us with deeper understanding of strictly k-testable languages.

Let $M_S = (Q, \Sigma, \delta, p_0, F)$ be the canonical DFA associated with S, exactly accepting $L = L(S)$. For each $q \in Q$, a string u is said to be a *k-leader* of q iff $lg(u) = k$ and there exists p_u in Q such that $\delta(p_u, u) = q$.

THEOREM 15. *Let L be a strictly k-testable language such that $L = L(S)$ for some $S = [A, B, C]$. Then,*

1) *L is k-reversible iff $A \cap B \cap C = \emptyset$,*
2) *a language $L - (A \cap B \cap C)$ is k-reversible.*

PROOF.

First, suppose that $A \cap B \cap C \neq \emptyset$. Then, there exists a string $w \in A \cap B \cap C$ such that $lg(w) = k$. Clearly w is in L. Further, we have that a string ww' is in L, where w' is the shortest string such that $R_k(ww') = w$. (Note that for no $z \in \Sigma^+$, $ww'z$ is in L.) Let $u_1 = z = \lambda$, $v = w$, and let u_2 be a prefix of ww' such that $u_2w = ww'$. Then, both $u_1vz(= w)$ and $u_2vz(= ww')$ are in L, where $lg(v) = k$. However, $u_1v\backslash L = w\backslash L \supseteq \{\lambda, w'\}$, while $u_2v\backslash L = ww'\backslash L = \{\lambda\}$, which implies that L is not k-reversible.

Conversely, suppose that $A \cap B \cap C = \emptyset$. Then, from the way of constructing the canonical DFA $M_S = (Q, \Sigma, \delta, p_0, F)$ associated with S where $A \cap B \cap C = \emptyset$, we observe that the k-leader of any state $[x]$ in Q is x itself. This implies that M_S satisfies the requirement of determinacy with lookahead k for k-reversibility [3]. Hence, M_S is a k-reversible acceptor and L is k-reversible.

Second, we may assume that $A \cap B \cap C \neq \emptyset$ and let $L' = L - (A \cap B \cap C)$. From the way of constructing the canonical DFA $M_S = (Q, \Sigma, \delta, p_0, F)$ associated with S where $A \cap B \cap C \neq \emptyset$, it is seen that for each $x \in A \cap B \cap C$, the existence of two states $[x]$ and $[\hat{x}]$ having a common k-leader x only violates the k-reversibility of M_S. Hence, for all $x \in A \cap B \cap C$, by deleting final states $[\hat{x}]$ from M_S, we obtain a DFA M_S' which satisfies the k-reversibility, and it holds that M_S' exactly accepts L'. Hence, L' is k-reversible. □

ACKNOWLEDGMENTS

A preliminary version of this paper appeared in the *Proceedings of 27th Hawaii International Conference on System Sciences*, Maui, Hawaii, Jan. 1994. This work is supported in part by Grants-in-Aid for Scientific Research No. 04229105 from the Ministry of Education, Science and Culture, Japan.

The authors are grateful to anonymous referees for their useful suggestions which greatly improved the draft of this paper.

REFERENCES

[1] N. Abe and H. Mamitsuka, "Prediction of Beta-Sheet Structures Using Stochastic Tree Grammars," *Proc Genome Informatics Workshop 5*, pp. 12–28, 1994.

[2] D. Angluin, "Inductive Inference of Formal Languages From Positive Data," *Information and Control*, vol. 45, pp. 117–135, 1980.

[3] D. Angluin, "Inference of Reversible Languages," *J. ACM*, vol. 29, pp. 741–765, 1982.

[4] D. Angluin and C.H. Smith, "Inductive Inference: Theory and Methods," *ACM Computing Surveys*, vol. 15, no. 3, pp. 237–269, 1983.

[5] S. Arikawa, S. Kuhara, S. Miyano, Y. Mukouchi, A. Shinohara, and T. Shinohara, "A Machine Discovery From Amino Acid Sequences by Decision Trees Over Regular Patterns," *New Generation Computing*, vol. 11, pp. 361–375, 1993.

[6] K. Asai, S. Hayamizu, and K. Onizuka, "Hmm With Protein Structure Grammar," *Proc. 26th Hawaii Int'l Conf. System Sciences*, pp. 783–791, 1993.

[7] J.A. Brzozowski and I. Simon, "Characterizations of Locally Testable Events," *Discrete Mathematics*, vol. 4, pp. 243–271, 1973.

[8] K. Culik II and T. Harju, "Dominoes and the Regularity of DNA Splicing Languages," K. Mehlhorn, ed., *Proc. ICALP '89*, pp. 222–233. New York: Springer-Verlag, 1989.

[9] H. Dayhoff and H. Calderone, "Composition of Proteins," *Altas of Protein Sequence and Structure*, vol. 5, no. 3, pp. 363–373, 1978.

[10] K.S. Fu and T.L. Booth, "Grammatical Inference: Introduction and Survey, Part 1 and 2," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 5, pp. 95–111 and 409–423, 1975.

[11] P. Garcia and E. Vidal, "Inference of k-Testable Languages in the Strict Sense and Application to Syntactic Pattern Recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, no. 9, pp. 920–925, Sept. 1990.

[12] P. Garcia, E. Vidal, and J. Oncina, "Learning Locally Testable Languages in the Strict Sense," *Algorithmic Learning Theory* (*Proc. First Int'l Workshop Algorithmic Learning Theory*), pp. 325–338. Ohmsha Ltd. and Springer, 1990.

[13] R.W. Gatterdam, "Splicing Systems and Regularity," *Int'l J. Computer Mathematics*, vol. 31, pp. 63–67, 1989.

[14] R.W. Gatterdam, "Algorithms for Splicing Systems," *SIAM J. Computing*, vol. 21, pp. 507–520, 1992.

[15] E.M. Gold, "Language Identification in the Limit," *Information and Control*, vol. 10, pp. 447–474, 1967.

[16] M. Gribskov, A.D. McLachlan, and D. Eisenberg, "Profile Analysis: Detection of Distantly Related Proteins," *Proc. Nat'l Academy Sciences USA*, vol. 84, pp. 4,355–4,358, 1987.

[17] M.A. Harrison, *Introduction to Formal Language Theory*. Reading, Mass.: Addison-Wesley, 1978.

[18] T. Head, "Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors," *Bull. Mathematical Biology*, vol. 49, pp. 737–759, 1987.

[19] T. Head, "Splicing Schemes and DNA," *Lindenmayer Systems*, G. Rozenberg and A. Salomma, eds., pp. 371–383. New York: Springer-Verlag, 1992.

[20] C. Helgesen and P.R. Sibbald, "Palm—A Pattern Language for Molecular Biology," *Proc. First Int'l Conf. Intelligent Systems for Molecular Biology*, pp. 172–180, 1993.

[21] J.E. Hopcroft, "An *n* log *n* Algorithm for Minimizing States in a Finite Automaton," *Theory of Machine and Computation*, A. Kohavi and A. Paz, eds., pp. 189–196, 1971.

[22] O.H. Ibarra and T. Jiang, "Learning Regular Languages From Counterexamples," *Proc. First Workshop on Computational Learning Theory*, pp. 337–351, 1988.

[23] S. Kobayashi and T. Yokomori, "Modeling RNA Secondary Structures Using Tree Grammars," *Proc. Fifth Genome Informatics Workshop*, Universal Academy Press, pp. 29–38, 1994.

[24] R. McNaughton and S. Papert, *Counter-Free Automata*. Cambridge, Mass.: MIT Press, 1971.

[25] S. Miyano, A. Shinohara, S. Arikawa, S. Shimozono, T. Shinohara, and S. Kuhara, "Knowledge Acquisition From Amino Acid Sequences by Decision Trees and Indexing," *Proc. Third Genome Informatics Workshop*, pp. 69–72, 1992.

[26] *Protein Database*. Osaka, Japan: Protein Research Foundation.

[27] Y. Sakakibara, M. Brown, R. Hughey, I.S. Mian, K. Sjolander, R.C. Underwood, and D. Haussler, "Stochastic Context-Free Grammars for tRNA Modeling," *Nucleic Acids Research*, vol. 22, pp. 5,112–5,120, 1994.

[28] D.B. Searls, "The Computational Linguistics of Biological Sequences," L. Hunter, ed., *Artificial Intelligence an Molecular Biology*, Chapter 2, pp. 47–120. AAAI Press, 1993.

[29] T. Shinohara, "Polynomial Time Inference of Extended Regular Pattern Languages," *Proc. RIMS Symp. Software Science and Eng.*, pp. 115–127. New York: Springer-Verlag, 1983.

[30] T. Shinohara, "Inductive Inference From Positive Data Is Powerful," *Proc. Third Workshop on Computational Learning Theory*, pp. 97–110, 1990.

[31] R. Siromoney, K.G. Subramanian, and V.R. Dare, "Circular DNA and Splicing Systems," *Proc. Int'l Conf. Parallel Image Analysis*, pp. 260–273. New York: Springer-Verlag, 1992.

[32] G.D. Stormo and G.W. Hartzell III, "Identifying Protein-Binding Sites From Unaligned DNA Fragments," *Proc. Nat'l Academy Sciences USA*, vol. 86, pp. 1,183–1,187, 1989.

[33] Y. Takada and R. Siromoney, "On Identifying DNA Splicing Systems From Examples," P.K. Jantke, ed., *Proc. AII '92*, pp. 305–319. New York: Springer-Verlag, 1992.

[34] N. Tanida and T. Yokomori, "Polynomial-Time Identification of Strictly Regular Languages in the Limit," *IEICE Trans. Information and Systems*, vol. 75-D, pp. 125–132, 1992.

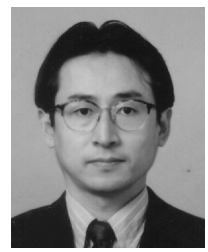
[35] J.D. Watson, J. Tooze, and D.T. Kurtz, *Recombinant DNA: A Short Course*. New York: Freeman, 1983.

[36] T. Yokomori, "On Polynomial-Time Learnability in the Limit of Strictly Deterministic Automata," *Machine Learning*, vol. 19, 1995.

[37] T. Yokomori and S. Kobayashi, "DNA Evolutionary Linguistics and RNA Structure Modeling: A Computational Approach," *Proc. IEEE Symp. Intelligence in Neural and Biological Systems*, pp. 38–45, 1995.

[38] T. Yokomori and S. Kobayashi, "On the Power of Circular Splicing Systems and DNA Computability," *Proc. IEEE Int'l Conf. Evolutionary Computation*, pp. 219–224, 1997.

Takashi Yokomori received the BS, MS, and PhD degrees from the University of Tokyo in 1974, 1976, and 1979, respectively. After working for the Department of Informatics at Sanno College and for IIAS-SIS, Fujitsu Limited, he joined the Department of Computer Science, University of Electro-Communications in 1989. Since April 1998, he has been a professor in the Department of Mathematics, Faculty of Education, Waseda University. He was a postdoctoral fellow at McMaster University, Canada, in 1981-1982 and was a visiting scholar at the University of Waterloo, Canada, in 1995-1996. His current research interests include formal language theory, computational learning theory, and bioinformatics. Dr. Yokomori is a member of the IEEE, ACM, EATCS, IEICE, IPSJ, and JSAI.



Satoshi Kobayashi received the BE, ME, and DE degrees from the University of Tokyo in 1988, 1990, and 1993, respectively, and joined the Department of Computer Science and Information Mathematics, University of Electro-Communications as a research associate in 1993. He has been an assistant professor at Tokyo Denki University since April 1998. His current research interests include computational learning theory, formal language theory, genome informatics, and molecular computation. Dr. Kobayashi is a member of the EATCS, IEICE, IPSJ, and JSAI.