INVITED PAPER *Special Issue on Algorithmic Learning Theory*

Algorithmic Learning Theory with Elementary Formal Systems

Setsuo ARIKAWA†, *Nonmember*, **Satoru MIYANO**†, *Member*,
Ayumi SHINOHARA†, **Takeshi SHINOHARA**††,
and **Akihiro YAMAMOTO**†††, *Nonmembers*

SUMMARY The elementary formal system (EFS, for short) is a kind of logic program which directly manipulates character strings. This paper outlines in brief the authors' studies on algorithmic learning theory developed in the framework of EFS's. We define two important classes of EFS's and a new hierarchy of various language classes. Then we discuss EFS's as logic programs. We show that EFS's form a good framework for inductive inference of languages by presenting model inference system for EFS's in Shapiro's sense. Using the framework we also show that inductive inference from positive data and PAC-learning are both much more powerful than they have been believed. We illustrate an application of our theoretical results to Molecular Biology.

key words *: algorithmic learning theory, computational learning theory, elementary formal system*

1. Introduction

The elementary formal system (EFS, for short) is a logical system introduced by Smullyan[38] in 1960 for describing the recursive function theory over character strings. The systems were shown to be a kind of grammars for generating formal languages like Chomsky grammars[4]. Smullyan expressed an axiom in a form $A_1 \rightarrow \cdots \rightarrow A_n \rightarrow B$, where $A_1, \cdots, A_n, B$ are atoms. This axiom is equivalent to a definite clause $B \leftarrow A_1, \cdots, A_n$. Hence, EFS is also a kind of logic program which directly manipulates character strings, and thus we can make use of the resolution principle[28] for recognizing formal languages.

Machine learning has been attracting much attention in Computer Science and Artificial Intelligence. From a lot of contributions to machine learning for more than 25 years[19]-[21] has emerged a new field of Theoretical Computer Science, that is, algorithmic learning theory. The algorithmic learning theory includes three major paradigms, inductive inference[3], learning with minimally adequate teacher (MAT)[2], and probably approximately correct (PAC) learning[39].

These learning paradigms have often been developed in terms of formal language theory so that they have dealt with various types of formal grammars and automata. These theories heavily depend on their own frameworks, and consequently it is sometimes difficult to compare the powers and efficiencies between different learning paradigms.

Most learning algorithms need procedures for testing the consistency or validity of guessed hypotheses. When we deal with, say regular expressions in inductive inference, we need to devise a procedure for converting a guessed regular expression into a finite automaton or something like that to test whether or not it is consistent with the examples given so far. And for context-free grammars, we need a procedure for making pushdown automata. In this way, we are forced to have special procedures depending on target expressions.

The algorithmic learning theory on formal languages can employ EFS's as a unifying framework. We can define a new hierarchy of various language classes with EFS's. In fact, it includes as a kernel the class of pattern languages that played an important role in inductive inference from positive data[1],[33],[34]. It also includes the four classes in Chomsky hierarchy, and many others[4],[7],[23],[29],[30],[37]. The resolution principle for EFS's works as a uniform procedure for testing guessed hypotheses. Thus EFS's work as grammars to generate languages, as automata to recognize languages, and as logic programs on character strings. In developing algorithmic learning theory with EFS's, we can take full advantage of the fruitful results accumulated in the fields above.

The present paper outlines in brief the authors' studies on algoritmic learning theory with EFS's[5]-[7],[23],[33],[37]. First we define the EFS's and important subclasses. Then, we discuss them from the aspects of logic programming. In Sect. 3 we describe model inference for EFS's in Shapiro's sense, and show that they form a good framework for inductive inference of formal languages. Using this framework, we

Manuscript received March 9, 1992.
Manuscript revised March 25, 1992.

† The authors are with the Research Institute of Fundamental Information Science, Kyushu University, Fukuoka-shi, 812 Japan.
†† The author is with the Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology, Iizuka-shi, 820 Japan.
††† The author is with the Faculty of Engineering, Hokkaido University, Sapporo-shi, 060 Japan.

show that inductive inference from positive data is much more powerful than it has been believed for many years. In Sect. 4, again due to our framework, we also show that EFS's have a powerful series of PAC-learnable classes. In Sect. 5 we illustrate an application of our theory with EFS's to Molecular Biology. Finally we refer to works on MAT-learning and learning by analogy developed in the framework of EFS's by the other researchers.

2. Elementary Formal Systems

Let Σ be a finite *alphabet*, X be a countable set of variables, and Π be a set of *predicate* symbols. We assume these three sets Σ, X, Π are mutually disjoint. The *arity*, a positive integer, is associated with each predicate symbol. We use x, y, x_1, x_2, $\cdots$ to denote variables and p, q, p_1, p_2, $\cdots$ to denote predicate symbols. By Σ^*, Σ^+, $\Sigma^{\leq n}$ we denote the sets of all strings over Σ, all nonempty strings, all strings of length n or less, respectively.

A *pattern* is an element in $(\Sigma \cup X)^+$. A pattern π is said to be *regular* if each variable appears at most once in π. For example, a pattern "xy" is regular but not "xx". An *atom* is an expression of the form $p(\pi_1, \cdots, \pi_n)$, where p is a predicate symbol with arity n and $\pi_1, \cdots, \pi_n$ are patterns. A *definite clause* is a clause of the form

$$A \leftarrow B_1, \cdots, B_m$$

where $A, B_1, \cdots, B_m$ are atoms. The atom A is called the *head* and the part $B_1, \cdots, B_m$ the *body* of the definite clause. We identify an atom A with a clause $A \leftarrow$. A clause is called *ground* if it contains no variable. The set of all ground atoms is called the *Herbrand base* and denoted by *HB*.

Definition 1: An *elementary formal system* (*EFS*, for short) is a finite set of definite clauses.

A *substitution* is a homomorphism from patterns to patterns that maps each symbol $a \in \Sigma$ to itself. A substitution that maps some variables to empty string is called an ε-substitution. In this paper we do not deal with the ε-substitution if stated otherwise. By $\pi\theta$, we denote the image of a pattern π by a substitution θ. For an atom $A = p(\pi_1, \cdots, \pi_n)$ and a clause $C = A \leftarrow B_1, \cdots, B_m$, we define $A\theta = p(\pi_1\theta, \cdots, \pi_n\theta)$ and $C\theta = A\theta \leftarrow B_1\theta, \cdots, B_m\theta$.

Definition 2: A definite clause C is *provable from* an EFS Γ, denoted by $\Gamma \vdash C$, if C is obtained by finitely many (possibly 0) applications of substitutions and modus ponens.

Definition 3: Let Γ be an EFS and p be a unary predicate symbol. The *language* $L(\Gamma, p)$ is the set $\{w \in \Sigma^+ | \Gamma \vdash p(w)\}$. If such Γ and p exist, the language L is *definable by* an EFS Γ or called an *EFS language*. The set of ground atoms provable from Γ is denoted by $PS(\Gamma)$.

The class of pattern languages was introduced by Angluin[1]. The language $L(\pi)$ of a pattern π is defined as the set of strings obtained by substituting nonempty strings for variables in π, that is, $L(\pi) = \{w \in \Sigma^+ | w = \pi\theta, \theta \text{ is a substitution}\}$. Here we should note that a pattern language $L(\pi)$ is an EFS language $L(\Gamma, p)$ with $\Gamma = \{p(\pi) \leftarrow\}$.

2.1 Subclasses of Elementary Formal Systems

We introduce several subclasses of EFS's, called variable-bounded EFS's, length-bounded EFS's, simple EFS's, regular EFS's, and linear EFS's, and show the relations of their languages to Chomsky hierarchy.

The set of variables contained in an atom A is denoted by $v(A)$. The length of a pattern π is denoted by $|\pi|$. For an atom, $|p(\pi_1, \cdots, \pi_n)| = |\pi_1| + \cdots + |\pi_n|$. The number of all occurrences of a variable x in a pattern π is denoted by $o(x, \pi)$. For an atom $o(x, p(\pi_1, \cdots, \pi_n)) = o(x, \pi_1) + \cdots + o(x, \pi_n)$.

Definition 4: A definite clause $A \leftarrow B_1, \cdots, B_m$ is said to be *variable-bounded* if

$$v(A) \supseteq v(B_1) \cup \cdots \cup v(B_m)$$

Definition 5: A definite clause $A \leftarrow B_1, \cdots, B_m$ is said to be *length-bounded* if

$$|A\theta| \geq |B_1\theta| + \cdots + |B_m\theta|$$

for any substitution θ.

We can easily characterize the length-boundedness as in the following lemma, from which we know that a length-bounded clause is variable-bounded and we can effectively determine if a given clause is length-bounded or not.

Lemma 1:[7] A definite clause $A \leftarrow B_1, \cdots, B_m$ is length-bounded if and only if $|A| \geq |B_1| + \cdots + |B_m|$ and $o(x, A) \geq o(x, B_1) + \cdots + o(x, B_m)$ for any variable x.

Definition 6: A *simple* clause is a clause of the form $p(\pi) \leftarrow q_1(x_1), \cdots, q_m(x_m)$, where $p, q_1, \cdots, q_m$ are unary predicate symbols and $x_1, \cdots, x_m$ are mutually distinct variables appearing in π.

Definition 7: A simple clause is called *regular* if the pattern in its head is regular.

Definition 8: A regular clause is called *right (left)-linear* if the pattern in the head is of the form $xw(wx)$ for some string $w \in \Sigma^*$.

An EFS Γ is called *variable-bounded* (resp. *length-bounded, simple, regular, right-linear, left-linear*) if all clauses in Γ are variable-bounded (resp. length-bounded, simple, regular, right-linear, left-linear).

Example 1:

(1) The context-sensitive language $\{a^n b^n c^n | n \geq 1\}$ is definable by a length-bounded EFS

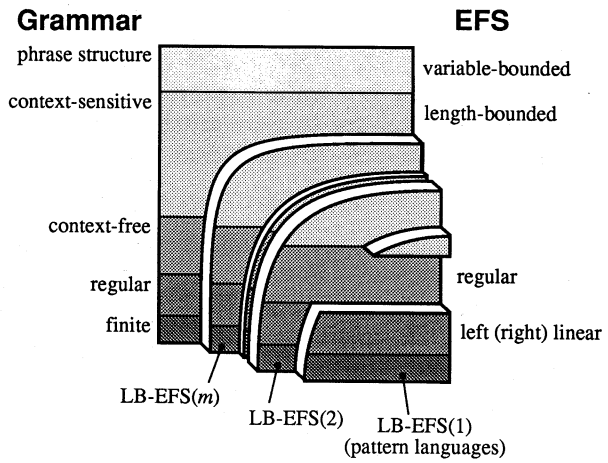


Fig. 1 Hierarchy of EFS languages. There is no pattern language which is context-free but not regular.

$$\Gamma_1=\left\{\begin{matrix} p(xyz)\leftarrow q(x,\ y,\ z) \\ q(ax,\ by,\ cz)\leftarrow q(x,\ y,\ z) \\ q(a,\ b,\ c) \end{matrix}\right\}$$

(2) The language $\{a^{2^n}|n\geqq 0\}$ is definable by a simple EFS

$$\Gamma_2=\left\{\begin{matrix} p(xx)\leftarrow p(x) \\ p(a) \end{matrix}\right\}$$

(3) The context-free language $\{a^n b^n|\ n\geqq 1\}$ is definable by a regular EFS

$$\Gamma_3=\left\{\begin{matrix} p(axb)\leftarrow p(x) \\ p(ab) \end{matrix}\right\}$$

For these classes of EFS languages, the following theorems hold.

Theorem 1:[7] A language is recursively enumerable (resp. context-sensitive, context-free, regular) if and only if it is definable by a variable-bounded (resp. length-bounded, regular, right/left-linear) EFS.

From the theorem, we can only deal with variable-bounded EFS's, that is, the variable-boundedness is not a restriction, because in this paper we are only interested in EFS's as language defining devices.

Theorem 2:[4],[29] The class of languages definable by simple EFS's is properly located between the classes of context-free languages and context-sensitive languages.

Let LB-EFS(m) be the class of all languages definable by length-bounded EFS's each of which has at most m clauses. Then together with the theorems above, we have a hierarchy in Fig. 1.

2.2 Elementary Formal Systems as Logic Programs

As seen in Theorem 1, EFS's are natural devices to generate formal languages. The EFS's can be considered as logic programs and a refutation procedure can be applied to the variable-bounded EFS's. Hence we can also consider EFS's as acceptors.

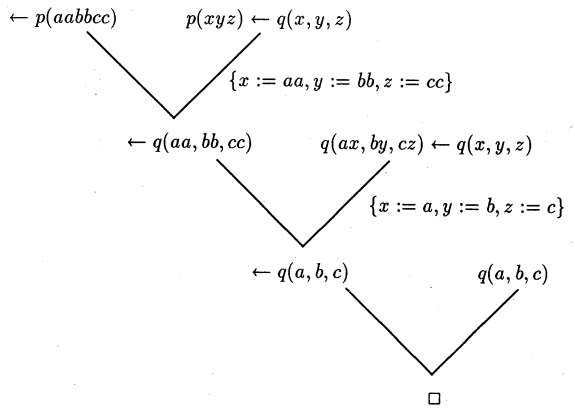


Fig. 2 Refutation of $\leftarrow p(aabbcc)$ in Γ_1.

Let α and β be patterns or atoms. Then a substitution θ is called *a unifier* of α and β if $\alpha\theta=\beta\theta$. If $\alpha=\beta\theta$ and $\alpha\theta'=\beta$ for some substitutions θ and θ', α is called a *variant* of β. In usual logic programming languages, any pair of terms or atoms has a unique most general unifier. However, there may be infinitely many maximally general unifiers for a pair of patterns. Let $\Sigma=\{a, b\}$. Then $\{x:=a^i\}$ for every i is the unifier of patterns ax and xa. All the unifiers are maximally general.

Hence, we need to define a new derivation for an EFS with no requirement that every unifier should be most general. A *goal* is a clause of the form $\leftarrow B_1, \cdots, B_m (m\geqq 0)$. We assume a *computation rule* R to select an atom from a goal.

Definition 9: Let Γ be an EFS, and G be a goal. A *derivation from* G is a (finite or infinite) sequence of triplets $(G_i, \theta_i, C_i)\ (i=0, 1, \cdots)$ that satisfies the following conditions:

(1) G_i is a goal, θ_i is a substitution, C_i is a variant of a clause in Γ, and $G_0=G$.

(2) $v(C_i)\cap v(C_j)=\phi\ \ (i\neq j)$, and $v(C_j)\cap v(G)=\phi$ for every i.

(3) If G_i is $\leftarrow A_1, \cdots, A_k$ and A_m is the atom selected by R, then C_i is $A\leftarrow B_1, \cdots, B_q$, and θ_i is a unifier of A and A_m, and G_{i+1} is

$$(\leftarrow A_1, \cdots, A_{m-1}, B_1, \cdots, B_q, A_{m+1}, \cdots, A_k)\,\theta_i$$

A_m is called the *selected atom* in G_i, and G_{i+1} is called a *resolvent* of G_i and C_i by θ_i.

Definition 10: A *refutation* is a finite derivation ending with empty goal $\square$.

Figure 2 depicts an example of a refutation in the EFS Γ_1 in Example 1.

Proposition 1: Let α and β be a pair of patterns or atoms. If one of them is ground, then every unifier of α and β is ground and the set of all unifiers is finite and computable, where a unifier $\{x_1:=t_1, \cdots, x_n:=t_n\}$ is

ground if all the $t_i (1 \leqq i \leqq n)$ are ground.

By this proposition, for variable-bounded EFS's, all goals in a derivation from a ground goal are kept ground. We can implement the derivation procedure in nearly the same way as in the traditional logic programming languages.

Now we describe the semantics of refutation according to Jaffar et al.[14] The unification in the EFS's can be considered as an equality theory E with an associative law.

The first semantics for an EFS Γ is its model. To interpret formulas we can restrict the domains to the models of E. Then

$$M(\Gamma) = \cap \{M \subset HB | M \text{ is an Herbrand model of } \Gamma\}$$

is an Herbrand model of Γ, and every ground atom in $M(\Gamma)$ is true in any model of Γ. The second semantics is the least fixed point $lfp(T_\Gamma)$ of the function $T_\Gamma : 2^{HB} \to 2^{HB}$ defined by

$$T_\Gamma(I) = \{A \in HB | \text{ there is a ground instance } A \leftarrow B_1, \cdots, B_n \text{ of a clause in } \Gamma \text{ such that } B_k \in I \text{ for } 1 \leqq k \leqq n\}$$

The third semantics using refutation is defined by

$$SS(\Gamma) = \{A \in HB | \text{there exists a refutation from } \leftarrow A\}$$

These three semantics are shown to be identical[14]. Hence we have the following theorem.

Theorem 3:[42] For every EFS Γ, $M(\Gamma) = lfp(T_\Gamma) = SS(\Gamma) = PS(\Gamma)$.

By this theorem the refutation is complete as a procedure of accepting EFS languages.

Now we discuss the inference of negation. In the traditional logic programming, the negation as failure rule is complete under the completion of definite programs, but it is not the case for EFS's.

A derivation is *finitely failed with length n* if its length is n and there is no clause which satisfies the condition (3) in Definition 9 for the selected atom in the last goal. A derivation (G_i, θ_i, C_i) $(i=0, 1, \cdots)$ is *fair* if it is finitely failed or, for each atom A in G_i, there is a $k \geqq i$ such that $A\theta_i \cdots \theta_{k-1}$ is the selected atom in G_k. We assume that any computation rule R makes all derivations *fair*. Such a computation rule is said to be *fair*.

The *negation as failure rule* is the rule that infers $\neg A$ when a ground atom A is in

$$FF(\Gamma) = \{A \in HB | \text{ for any fair computation rule, there is an integer } n \text{ such that all derivations from } \leftarrow A \text{ are finitely failed within length } n\}$$

Put $E(\theta) = (x_1 = \tau_1 \wedge \cdots \wedge x_n = \tau_n)$ for a substitution $\theta = \{x_1 := \tau_1, \cdots, x_n := \tau_n\}$, and $E(\theta) =$ true for an identity substitution θ. Negation as failure for EFS's is complete if the following two are satisfied[14]:

(4) There is a theory E^* such that, for every two patterns π and τ, $(\pi = \tau) \to \bigvee_{i=1}^{k} E(\theta_i)$ is a logical consequence, where $\theta_1, \cdots, \theta_k$ are all unifiers of π and τ, and the disjunction means $\square$ if $k=0$.

(5) $FF(\Gamma)$ is identical to the set

$$GF(\Gamma) = \{A \in HB | \text{ for any fair computation rule, all derivations from } \leftarrow A \text{ are finitely failed}\}$$

In case of traditional logic program, it is known that $FF(\Gamma) = GF(\Gamma)$ since the most general unifier is uniquely determined[17]. However, we can easily construct an EFS Γ such that $FF(\Gamma) \neq GF(\Gamma)$ because a pair of patterns may have infinitely many maximally general unifiers. Note that such an EFS Γ may not be variable-bounded.

We show that negation as failure rule for variable-bounded EFS is complete, for which we need the set

$$GGF(\Gamma) = \{A \in HB | \text{ for any fair computation rule, all derivations from } \leftarrow A \text{ such that all goals in them are ground are finitely failed}\}$$

The inference rule that infers $\neg A$ for a ground atom A if A is not in $GGF(\Gamma)$ is called the *Herbrand rule*[17].

Theorem 4:[42] For any variable-bounded EFS Γ,

$$FF(\Gamma) = GF(\Gamma) = GGF(\Gamma)$$

Thus the negation as failure is complete and identical to the Herbrand rule for variable-bounded EFS's. Hence we have the following theorem.

Theorem 5:[42] Let Γ be a variable-bounded EFS and A be a ground atom. Then A is provable from Γ if and only if there is a refutation from $\leftarrow A$ in Γ.

3. Inductive Inference with Elementary Formal Systems

First, we discuss the theory of inductive inference developed with EFS's. Inductive inference is one of the mathematical models of algorithmic learning. Identification in the limit is a notion of successful inference introduced by Gold[12].

3. 1 Identification in the Limit

Let U be a set, which we call a *universal set*. A *rule* is a subset $R \subseteqq U$. An *indexed family of recursive*

rules is a class of rules $C=R_1, R_2, \cdots$ for which there exists a computable function $f: N\times U\rightarrow\{0, 1\}$ such that $f(i, s)=1$ or 0 if s is in R_i or not, respectively.

For EFS languages, the universal set U is the set Σ^+. For EFS models, U is the set of all ground atoms. We can consider an EFS as the index of a rule. For any length-bounded EFS Γ, any predicate symbol p and any string w, we can effectively determine whether $p(w)$ is provable from Γ. Therefore, models or languages definable by a length-bounded EFS constitute an indexed family of recursive rules.

A *complete presentation* of a rule R is an infinite sequence $(s_1, t_1), (s_2, t_2), \cdots$ such that t_i is 0 or 1, $\{s_i|t_i=1\}=R$ and $\{s_i|t_i=0\}=U-R$. A *positive presentation* of a nonempty rule R is an infinite sequence $s_1, s_2, \cdots$ such that $\{s_i|i=1, 2, \cdots\}=R$.

An *inference machine* is an effective procedure that requests input and produces output from time to time. We call an output produced by an inference machine a *guess*. Let $\sigma=s_1, s_2, \cdots$ be an infinite sequence and $g_1, g_2, \cdots$ be the sequence of guesses produced by an inference machine M when elements in σ are successively given to M. Then we say that M *on input* σ *converges to* g, if all but finitely many guesses out of $g_1, g_2, \cdots$ are equal to g.

Definition 11: A class of rules $C=R_1, R_2, \cdots$ is said to be *inferable from complete* (resp. *positive*) *data* if there exists an inference machine M such that for any index i and any complete (resp. positive) presentation of R_i, M on input σ converges to g with $R_g=R_i$. We also say that the machine M identifies C in the limit from complete (resp. positive) data.

3.2 Model Inference System for Elementary Formal Systems

Model inference system (MIS, for short) developed by Shapiro[32] is a kind of automatic program synthesis system for Prolog programs. From other systems, MIS is distinguished by fully utilizing the tight relationship between syntax and semantics of logic programs. Since the variable-bounded EFS's have nearly the same properties as the usual logic programs as we have seen in the previous section, we can naturally consider a learning algorithm for them based on the Shapiro's theory of model inference.

In model inference, examples are given as a complete presentation of the model of an EFS. Therefore, positive examples of an EFS Γ are ground atoms provable from Γ, and negative examples are the rest. A hypothesis H is said to be *too strong* if H proves some negative examples. H is said to be *too weak* if H fails to prove some positive examples.

The following procedure MIS outlines the model inference system.

```
Procedure MIS;
begin
    H := {□};
    repeat
        read next example;
        while H is too strong or too weak do
        begin
            while H is too strong do
            begin
                apply CBA to H and detect a false clause C in H;
                remove C from H;
            end;
            while H is too weak do
                add a refinement of a clause removed
        end;
        output H;
    forever
end;
```

In order for MIS to correctly infer EFS models, we devise two algorithms, the contradiction backtracing algorithm CBA and the refinement operator.

Given a refutation claiming that a hypothesis H is too strong, CBA finds a false clause from H by tracing selected atoms backward. To test the truth value of a selected atom, CBA calls an oracle ASK that receives a ground atom and returns its truth value. We can easily simulate ASK since examples are given as a complete presentation. However, CBA must take a ground instance of a tested atom that is not ground, since ASK can answer only ground atoms. Fortunately, all the selected atoms in a refutation from a ground goal in a variable-bounded EFS are ground. Hence we can simplify CBA for variable-bounded EFS's as follows.

```
Procedure CBA_for_EFS;
input:  (G_0 = G, θ_0, C_0), (G_1, θ_1, C_1), ..., (G_k = □, θ_k, C_k);
        {a refutation of a ground goal G false in M}
output: a clause C_i false in M;
begin
    for i := k downto 1 do begin
        let A_i be the selected atom of G_{i-1};
        if ASK(A_i) is false then return C_{i-1};
    end
end
```

Theorem 6:[7] Let M be a model of a variable-bounded EFS Γ, and $(G_0=G, \theta_0, C_0), (G_1, \theta_1, C_1), \cdots, (G_k=\square, C_k, \theta_k)$ be a refutation by Γ of a ground goal G true in M. If CBA is given the refutation, then it makes i oracle calls and returns C_{i-1} false in M for some $i=1, 2, \cdots, k$.

Now we consider the refinement operator. First we need the notion of *size*.

Definition 12: We define the size of an atom A by

size$(A)=2\times|A|-\#\, v(A)$, and size$(C)=2\times(|A|+|B_1|+\cdots+|B_n|)-\#\, v(C)$ for a clause $C=A\leftarrow B_1, \cdots, B_n$.

For a binary relation R, $R(a)$ denotes the set $\{b \mid (a, b)\in R\}$ and R^* denotes the reflexive transitive closure of R. A clause D is called a *refinement* of C if D is a logical consequence of C and size$(C)<$ size(D). A *refinement operator* ρ is a subrelation of refinement relation such that the set $\{D\in\rho(C) \mid \text{size}(D)\leqq n\}$ is finite and computable. A refinement operator ρ is *complete for a set* S if $\rho^*(\square)=S$. A refinement operator ρ is *locally finite* if $\rho(C)$ is finite for any clause C.

Now we introduce refinement operators for the classes of EFS's. All refinement operators defined below have a common feature. They are constructed by two types of operations, application of a substitution and addition of an atom.

Definition 13: A substitution θ is *basic for a clause* C if

(1) $\theta=\{x:=y\}$, where $x\in v(C)$, $y\in v(C)$ and $x\neq y$,
(2) $\theta=\{x:=a\}$, where $x\in v(C)$ and $a\in\Sigma$, or
(3) $\theta=\{x:=yz\}$, where $x\in v(C)$, $y\notin v(C)$, $z\notin v(C)$ and $y\neq z$.

Definition 14: Let A be an atom. Then an atom B is in $\rho_a(A)$ if and only if

(4) $A=\square$ and $B=p(x_1, \cdots, x_n)$ for $p\in\Pi$ with arity n and mutually distinct variables $x_1, \cdots, x_n$, or
(5) $A\theta=B$ for a substitution θ basic for A.

For ρ_a we have the following completeness theorem.

Theorem 7:[7] ρ_a is a locally finite and complete refinement operator for atoms.

Definition 15: Let C be a variable-bounded clause. Then a clause D is in $\rho_{vb}(C)$ if and only if (4) or (5) in Definition 14 holds, or $C=A\leftarrow B_1, \cdots, B_{n-1}$ and $D=A\leftarrow B_1, \cdots, B_{n-1}, B_n$ is variable-bounded. Similarly we define ρ_{lb} for length-bounded clauses.

Theorem 8:[7]

(1) ρ_{vb} is a complete refinement operator for variable-bounded clauses.
(2) ρ_{lb} is a locally finite and complete refinement operator for length-bounded clauses.

Note that ρ_{vb} is not locally finite because the number of atoms B_n possibly added by ρ_{vb} is infinite, while ρ_{lb} is locally finite. We can also define refinement operators for simple or regular clauses which are locally finite and complete. For simple clauses, applications of the basic substitutions are restricted only to clauses without any bodies, and for regular clauses, substitutions of the form $\{x:=y\}$ are not allowed.

3.3 Inductive Inference from Positive Data

Gold[12] showed that any indexed family of recursive languages is inferable from complete data while it is not always inferable from positive data. For example, even the class of regular languages can not be identified in the limit from positive data.

However, Angluin[1] showed some interesting classes inferable from positive data such as pattern languages. Shinohara[35] and Wright[41] showed that the class consisting of unions of pattern languages is inferable from positive data. Shinohara[36] proved the inferability of languages from positive data, when they are definable by a simple EFS with just two clauses.

More explicitly exploiting the framework of EFS's, we can reveal the existence of rich classes inferable from positive data. Since we deals with EFS languages, we can assume that every EFS contains a unary predicate symbol p and simply denote $L(\Gamma, p)$ by $L(\Gamma)$ without any loss of generality.

Definition 16: An EFS Γ is *reduced with respect to* a set $S\subseteqq\Sigma^*$ if $S\subseteqq L(\Gamma)$ but $S\nsubseteqq L(\Gamma')$ for any $\Gamma'\subsetneqq\Gamma$.

The key property that many classes of length-bounded EFS's are inferable from positive data is given in the following lemma. An EFS Γ_1 is said to be *equivalent* to Γ_2 if we can identify Γ_1 and Γ_2 up to renaming of variables and predicate symbols. Clearly, if Γ_1 is equivalent to Γ_2, then $L(\Gamma_1)=L(\Gamma_2)$.

Lemma 2:[37] For any $m\geqq 0$ and any finite set S of strings, there exist only finitely many inequivalent length-bounded EFS's with at most m clauses that are reduced with respect to S.

In the context of PAC-learning, we will again count the number of reduced EFS's more precisely. By using the lemma above and the notion of finite elasticity, which is a sufficient condition for inferability from positive data[41], we have the following theorem.

Theorem 9:[37] The class LB-EFS(m) is inferable from positive data.

Since we can designate the m arbitrarily, the classes inferable from positive data unlimitedly increase as shown in Fig. 1.

4. PAC-Learnable Elementary Formal Systems

4.1 Probably Approximately Correct Learning

For an alphabet Σ, a subset c of Σ^* is called a *concept*. Obviously, we can regard a concept c as its characteristic function $c:\Sigma^*\rightarrow\{0, 1\}$ defined by $c(x)=1(x\in c)$, $c(x)=0$ $(x\notin c)$. A *concept class* is a nonempty set $\mathscr{C}\subseteqq 2^{\Sigma^*}$ of concepts. For a concept c, an *example* of c is a pair $\langle x, c(x)\rangle$ for $x\in\Sigma^*$. An example $\langle x, c(x)\rangle$ is said to be *positive* (resp. *negative*) if $c(x)=1$ (resp. $c(x)=0$). We say that a concept c is consistent with examples $\langle x_1, a_1\rangle, \langle x_2, a_2\rangle, \cdots, \langle x_n, a_n\rangle$ if $c(x_i)=a_i$ for all $1\leqq i\leqq n$. For a concept class $\mathscr{C}$, we assume a representation system of concepts in $\mathscr{C}$. For example, the class of regular sets has finite automata as representations of its concepts. A concept

need not be uniquely represented.

The following notion of PAC-learnability is due to[10],[25].

Definition 17: A concept class $\mathscr{C}$ is *polynomial-time learnable* if there exists an algorithm $\mathscr{A}$ which satisfies the following conditions:

(1) $\mathscr{A}$ runs in polynomial-time with respect to the input length.

(2) There is a polynomial $p(\cdot, \cdot, \cdot)$ such that for any integer $n \geqq 0$, any concept $c \in \mathscr{C}$, any real numbers ε, $\delta (0 < \varepsilon, \delta < 1)$, and any probability distribution P on $\Sigma^{\leqq n}$, if $\mathscr{A}$ takes $p(n, 1/\varepsilon, 1/\delta)$ examples which are generated randomly according to P, then $\mathscr{A}$ outputs a representation of a hypothesis h such that $P(c \oplus h) < \varepsilon$ with probability at least $1-\delta$, where $\oplus$ means the symmetric difference.

Definition 18:[24),(25)] For a concept $c \in \mathscr{C}$ and an integer $n \geqq 0$, we define $\dim \mathscr{C}_n = \log_2 |\mathscr{C}_n|$, where $\mathscr{C}_n = \{c \cap \Sigma^{\leqq n} | c \in \mathscr{C}\}$. We say that a concept class $\mathscr{C}$ is of *polynomial dimension* if there exists a polynomial $d(n)$ such that $\dim \mathscr{C}_n \leqq d(n)$ for all $n \geqq 0$.

Example 2: The class of all finite subsets of Σ^* is not of polynomial dimension. In the same way, the class of all regular sets is not of polynomial dimension.

Definition 19:[10] Let $\mathscr{C}$ be a concept class. A *randomized polynomial-time hypothesis finder for* $\mathscr{C}$ is a randomized polynomial-time algorithm that takes a sequence of examples of a concept in $\mathscr{C}$ as input, and produces a hypothesis in $\mathscr{C}$ that is consistent with the examples, with probability at least γ for some $\gamma > 0$. A *polynomial-time hypothesis finder for* $\mathscr{C}$ is a deterministic polynomial-time algorithm which finds a hypothesis consistent with the examples.

The polynomial-time learnability is characterized as follows:

Lemma 3:[10),(25)] Let $\mathscr{C}$ be a concept class. Consider the following conditions (a)-(c):

(a) $\mathscr{C}$ is of polynomial dimension.

(b) There is a polynomial-time hypothesis finder for $\mathscr{C}$.

(c) There is a randomized polynomial-time hypothesis finder for $\mathscr{C}$.

Then the following statements hold:

(1) If $\mathscr{C}$ is polynomial-time learnable, then (a) and (c) hold.

(2) If (a) and (b) hold, then $\mathscr{C}$ is polynomial-time learnable.

4.2 Polynomial-Time Learnable Classes

Lemma 3 (1) asserts that a polynomial-time learnable concept class $\mathscr{C}$ must be of polynomial dimension. We have the following theorem:

Theorem 10:[23] LB-EFS(m) is of polynomial dimension for any $m \geqq 1$.

The class LB-EFS of all length-bounded EFS languages contains all finite sets. Therefore, in order to obtain a polynomial-time learnable concept class of length-bounded EFS languages, the number of definite clauses must be bounded by a constant even if no variables appear in definite clauses.

Although LB-EFS(m) is of polynomial dimension, this class is not polynomial-time learnable under a reasonable assumption. We discuss this matter in the sequel. We consider the following class of length-bounded EFS's.

Definition 20: A definite clause

$$q(\pi_1, \cdots, \pi_n) \leftarrow q_1(\tau_1, \cdots, \tau_{t_1}), q_2(\tau_{t_1+1}, \cdots, \tau_{t_2}), \cdots, q_l(\tau_{t_{l-1}+1}, \cdots, \tau_{t_l})$$

is said to be *hereditary* if, for each $j = 1, \cdots, t_l$, pattern τ_j is a substring of some π_i. We say that an EFS Γ is *hereditary* if each definite clause in Γ is hereditary.

All languages in Example 1 are defined by length-bounded hereditary elementary formal systems. By definition, a simple EFS is also hereditary.

Definition 21: For $m, k \geqq 1$, we denote by LB-H-EFS(m, k) the class of languages definable by length-bounded hereditary EFS's with at most m definite clauses such that the number of variable occurrences in the head of each clause is bounded by k and the number of atoms in the body is also bounded by k.

Obviously the class LB-H-EFS(m, k) contains infinitely many languages for any m and k. Furthermore, $\bigcup_{m \geqq 1}$ LB-H-EFS$(m, 2)$ contains all context-free languages and $\bigcup_{m \geqq 1}$ LB-H-EFS$(m, 1)$ contains all linear context-free languages, therefore, all regular languages. Thus we can say that LB-H-EFS(m, k) is large enough when m is appropriately chosen.

Since LB-H-EFS(m, k) is a subclass of LB-EFS(m), it is of polynomial dimension. By Lemma 3 (2), we proved the following theorem by showing a polynomial-time hypothesis finder for LB-H-EFS(m, k).

Theorem 11:[23] LB-H-EFS(m, k) is polynomial-time learnable for any $m, k \geqq 1$.

Vitter and Lin[40] introduced the notion of **NC**-learnability by employing **NC** algorithms instead of polynomial-time algorithms. We also proved the following theorem:

Theorem 12:[23] LB-H-EFS(m, k) is $\mathbf{NC}^2$-learnable for any $m, k \geqq 1$.

4.3 Classes Hard to Learn in Polynomial Time

Any subclass $\mathscr{C}$ of LB-EFS(m) is of polynomial dimension. Therefore, if we can find a polynomial-time hypothesis finder for $\mathscr{C}$, it is a polynomial-time learning algorithm satisfying the conditions of Definition 17. We put a bound on the number of variable occurrences in defining the class LB-H-EFS(m, k). This

section provides a reason why this restriction is necessary to obtain polynomial-time learnable classes.

For a concept class $\mathcal{C}$, we consider the following problem:

Consistency Problem for $\mathcal{C}$

Instance: Finite sets P, $N \subseteq \Sigma^*$.

Question: Is there a concept $h \in \mathcal{C}$ consistent with the positive examples in P and the negative examples in N, i.e., $P \subseteq h$ and $N \subseteq \Sigma^* - h$?

If the consistency problem for $\mathcal{C}$ is **NP**-hard, it can be shown that $\mathcal{C}$ is not polynomial-time learnable under the assumption **RP**$\neq$**NP**. Ko and Tzeng[16] showed that the consistency problem for the class of pattern languages is Σ_2^p-complete. Schapire[31] also showed that the class of pattern languages is not polynomial-time learnable regardless of the representation under some reasonable assumption. Thus even LB-EFS(1) is hard to be polynomial-time learnable.

The following result asserts that the polynomial-time learnability requires a constant bound on the number of variable occurrences even for the class of regular pattern languages.

Theorem 13:[23] The consistency problem for the class of regular pattern languages is **NP**-complete.

By allowing ε-substitutions to a regular pattern π, we define a language $\tilde{L}(\pi)$ called an *extended regular pattern language*[34]. The problem of finding a sequence $a_1 \cdots a_n$ which is common to all positive examples but not common to any of negative examples is equivalent to finding a regular pattern $\pi = x_0 a_1 x_1 \cdots x_{n-1} a_n x_n$ such that the extended regular pattern language $\tilde{L}(\pi)$ is consistent with the positive and negative examples. This is slightly different from the longest common subsequence problem that is shown **NP**-complete[18]. The *common subsequence* can be expressed as a regular pattern of the form $x_0 a_1 x_1 a_2 x_2 \cdots x_{n-1} a_n x_n$ with $a_i \in \Sigma$ for $1 \leq i \leq n$.

As to this matter, we have the following results that also imply the hardness of polynomial-time learnability.

Theorem 14:[23]

(1) The consistency problem for the class of extended regular pattern languages is **NP**-complete.

(2) The consistency problem for the class of common subsequence languages is **NP**-complete.

5. Application to Molecular Biology

It is important to combine theory and pragmatism for the sound development of algorithmic learning theory. We present an application of the theory introduced in the former section.

We have applied learning algorithms to knowledge acquisition from amino acid sequences and shown that these approaches are very successful[5),(6)]. This section shows a method which employs a learning algorithm for EFS's by following Ref.(5). The problem we considered in Ref.(5) is to identify transmembrane domains in proteins from their amino acid sequences.

Regular patterns have been used to describe some features of amino acid sequences in PROSITE database[9]. For example, the zinc-finger motif is described as $x_1 C x_2 C x_3 H x_4 H x_5$ with some length constraints on x_2, x_3, x_4. We view amino acid sequences through such regular patterns.

The class LB-H-EFS(m, k) is polynomial-time learnable in the sense of PAC-learning (Theorem 11). Although the learning algorithm for Theorem 11 runs in polynomialtime, it requires a large amount of time and space, and cannot be used directly in practical applications.

In order to make the computation feasible, we restrict our attention to the class of languages definable by EFS's of the form

$$\{p(\pi_1), p(\pi_2), \cdots, p(\pi_n)\}$$

where $n \leq m$ and $\pi_1, \pi_2, \cdots, \pi_n$ are regular patterns with at most k variables. In other words, a language in this class is a union of at most m languages defined by regular patterns with at most k variables.

Furthermore, instead of applying the exact polynomial-time learning algorithm, we devised an algorithm (procedure find_union) which finds, from given sets P and N of positive and negative training examples, an EFS covering P and excluding N.

```
procedure find_union( P, N: strings );
begin
    S ← ∅;
    foreach pattern π with πθ = w
            for some w ∈ P and some substitution θ;
        if L(π) ∩ N = ∅ then S ← S ∪ {π};
    Find a subset Γ of S covering P which is minimal
        with respect to set-inclusion;
    output Γ;
end;
```

The procedure find_union(P, N) produces a collection of regular patterns made from P which covers P and excludes N.

In procedure find_union we employ the greedy approximation algorithm for the minimum set cover problem by Johnson[15]. This greedy algorithm has been shown to find a set cover of size at most $M \log M$ in polynomial time, where M is the size of the minimum set cover. Hence procedure find_union may not produce the smallest hypothesis, but it is guaranteed to construct a small enough hypothesis.

Assuming that the sequences corresponding to transmembrane domains of proteins are describable by an EFS of the above form with small m and k, we made experiments using PIR database[26] and obtained very acceptable hypotheses.

6. Discussion

We have overviewed in brief our studies on algorithmic learning theory in the framework of EFS's, and shown that the framework is convenient for studying inductive inference and PAC-learning. There remain another main paradigms in this particular field, MAT-learning and learning by analogy.

As for MAT-learning, Sakakibara[30] introduced an interesting subclass of variable-bounded EFS's called extended simple EFS's by generalizing the simple EFS's in Ref. (4), and proved that the class can be MAT-learnable via equivalence and membership queries plus some additional information. He also presented some results on classes of EFS's from the viewpoint of formal language theory. As for learning by analogy, Miyahara[22] discussed analogical reasoning using EFS's with mismatch in the sense of string pattern matching. His study was motivated by genome knowledge acquisition and based on the theory of analogical reasoning by Haraguchi and Arikawa[13].

Apart from algorithmic learning, we can easily define many extensions of context-free grammars in a very natural way. Saeki and Arikawa[29] proved that the Earley's parsing algorithm[11] can be extended to such classes of EFS's which include context-free grammars. EFS's have aspects of logic programming as we have seen. So we can make good use of results on logic programming. Conversely the results on EFS's are reflected into the usual logic programming as in Arimura[8].

References

(1) Angluin D.: "Finding patterns common to a set of strings", Proc. 11th ACM Symposium on Theory of Computing, pp. 130-141 (1979).

(2) Angluin D.: "Queries and concept learning", Machine Learning, **2**, pp. 319-342 (1988).

(3) Angluin D. and Smith C. H.: "Inductive inference: theory and methods", ACM Computing Surveys, **3**, pp. 237-269 (1983).

(4) Arikawa S.: "Elementary formal systems and formal languages — Simple formal systems", Memoirs of Faculty of Science, Kyushu University, Ser. A., Mathematics, **24**, pp. 47-75 (1970).

(5) Arikawa S., Kuhara S., Miyano S., Shinohara A. and Shinohara T.: "A learning algorithm for elementary formal systems and its experiments on identification of transmembrane domains", Proc. 25th Hawaii Int. Conf. System Sciences, pp. 675-684 (1992).

(6) Arikawa S., Kuhara S., Miyano S., Mukouchi Y., Shinohara A. and Shinohara T.: "A machine discovery from amino acid sequences by decision trees over regular patterns", Proc. Fifth Generation Computing Systems 1992, pp. 618-625 (1992).

(7) Arikawa S., Shinohara T. and Yamamoto A.: "Elementary formal system as a unifying framework for language learning", Proc. 2nd Workshop on Computational Learning Theory, pp. 312-32 (1989). (also in *Theoretical Computer Science*, **95**, pp. 97-113 (1992)).

(8) Arimura H.: "Completeness of depth-bounded resolution for weakly reducing programs", RIFIS-TR-CS-21, Research Institute of Fundamental Information Science, Kyushu University (1990).

(9) Bairoch A.: "PROSITE: a dictionary of sites and patterns in proteins", Nucleic Acids Res., **19**, pp. 2241-2245 (1991).

(10) Blumer A., Ehrenfeucht A., Haussler D. and Warmuth M. K.: "Learnability and the Vapnik-Chervonenkis dimension", J. ACM, **36**, 929-965 (1989).

(11) Earley J.: "An efficient context-free parsing algorithm", Commun. ACM, **13**, pp. 94-102 (1970).

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

(13) Haraguchi M. and Arikawa S.: "A formulation of reasoning by analogy: analogical union of logic programs", Lecture Notes in Computer Science, **264**, pp. 58-69 (1987).

(14) Jaffar J., Lassez J. L. and Mahr M. J.: "Logic programming scheme", Logic Programming: Functions, Relations, and Equations, pp. 211-233 (1986).

(15) Johnson D. S.: "Approximation algorithms for combinatorial problems", J. Comput. System Sci., **9**, pp. 256-278 (1974).

(16) Ko K. and Tzeng W.: "Three Σ_2^p-complete problems in computational learning theory", preprint (1990).

(17) Lloyd J. W.: "Foundations of Logic Programming, Second, Extended Edition", Springer-Verlag (1987).

(18) Mair D.: "The complexity of some problems on subsequences and supersequences", J. ACM, **25**, pp. 322-336 (1978).

(19) Michalski R. S., Carbonell J. G. and Mitchell T. M. (eds): "Machine Learning: An Artificial Intelligence Approach, Vol. I", Tioga Publishing Company (1983).

(20) Michalski R. S., Carbonell J. G. and Mitchell T. M. (eds): "Machine Learning: An Artificial Intelligence Approach, Vol. II", Morgan Kaufmann (1986).

(21) Michalski R. S., Kodratoff Y. (eds): "Machine Learning: An Artificial Intelligence Approach, Vol. III", Morgan Kaufmann (1990).

(22) Miyahara T.: "Analogical reasoning using elementary formal system with mismatch", Proc. 2nd Workshop on Algorithmic Learning Theory, pp. 224-230 (1991).

(23) Miyano S., Shinohara A. and Shinohara T.: "Which classes of elementary formal systems are polynomial-time learnable?", Proc. 2nd Workshop on Algorithmic Learning Theory, pp. 139-150 (1991).

(24) Natarajan B. K.: "On learning boolean functions", Proc. 19th ACM Symposium on Theory of Computing, pp. 296-304 (1987).

(25) Natarajan B. K.: "On learning sets and functions", Machine Learning, **4**, pp. 67-97 (1989).

(26) Protein Identification Resource, National Biomedical Research Foundation.

(27) Plotkin G. D.: "Building in equational theories", Machine Intelligence, **7**, pp. 132-147 (1972).

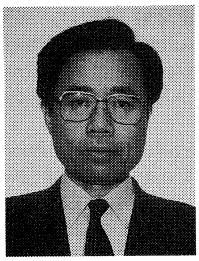
(28) Robinson J. A.: "A machine-oriented logic based on the resolution principle", J. ACM, **12**, pp. 23-41 (1965)

(29) Saeki I. and Arikawa S.: "Polynomial time parsers for elementary formal systems", SIG-FAI-9001-6, pp. 55-64 (1990).

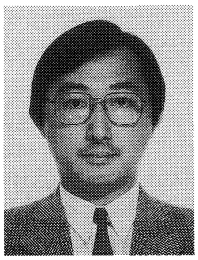
(30) Sakakibara Y.: "On learning Smullyan's elementary formal systems: towards an efficient learning method for context-sensitive languages", Advances in Software Science and Technology, **2**,pp. 79-101 (1990).

(31) Schapire R. E.: "Pattern languages are not learnable",

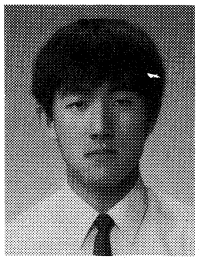
Proc. 3rd Workshop on Computational Learning Theory, pp. 122-129 (1990).
(32) Shapiro E.: "Inductive inference of theories from facts", Technical Report 192, Department of Computer Science, Yale University (1981).
(33) Shinohara T.: "Polynomial time inference of pattern languages and its applications", Proc. 7th IBM Symp. on Mathematical Foundations of Computer Science, pp. 191-209 (1982).
(34) Shinohara T.: "Polynomial time inference of extended regular pattern languages", Proc. RIMS Symposia on Software Science and Engineering, (Lecture Notes in Computer Science, **147**), pp. 115-127 (1983).
(35) Shinohara T.: "Inferring unions of two pattern languages", Bull. Inf. Cybern., **20**, pp. 83-88 (1983).
(36) Shinohara T.: "Inductive inference of formal systems from positive data", Bull. Inf. Cybern., **22**, pp. 9-18 (1986).
(37) Shinohara T.: "Inductive inference from positive data is powerful", Proc. 3rd Workshop on Computational Learning Theory, pp. 97-110 (1990). (to appear in Information and Computation)
(38) Smullyan R. M.: "Theory of Formal Systems", Princeton University Press (1961).
(39) Valiant L.: "A theory of the learnable", Commun. ACM, **27**, pp. 1134-1142 (1984).
(40) Vitter J. S. and Lin J.: "Learning in parallel", Proc. 1st Workshop on Computational Learning Theory, pp. 83-96 (1988).
(41) Wright K.: "Identification of unions of languages drawn from an identifiable class", Proc. 2nd Workshop on Computational Learning Theory, pp. 328-333 (1989).
(42) Yamamoto A.: "Elementary formal system as a logic programming language", Proc. Logic Programming Conference '89, pp. 123-132(1989). (also in J. Logic Programming, **13**, pp. 89-97 (1992))



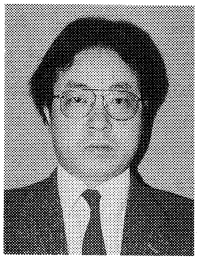
Setsuo Arikawa was born in Kagoshima, Japan in 1941. He received the B.S. degree in 1964, the M.S. degree in 1966 and the Dr. Sci. degree in 1969 all in Mathematics from Kyushu University, Fukuoka, Japan. Presently, he is a Professor of Research Institute of Fundamental Information Science, Kyushu University. His research interests include algorithmic learning theory, logic and inference in AI, and information retrieval systems.



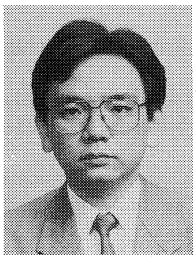
Satoru Miyano was born in Oita, Japan in 1954. He received the B.S. degree in 1974, the M.S. degree in 1979 and the Dr.Sci. degree in 1984 all in Mathematics from Kyushu University, Fukuoka, Japan. Currently, he is an Associate Professor of Research Institute of Fundamental Information Science, Kyushu University. His present interests are parallel algorithms, computational complexity, algorithmic learning theory and bioinformatics.



Ayumi Shinohara was born in Fukuoka, Japan in 1965. He received the B.S. degree in 1988 in Mathematics and M.S. degree in 1990 in Information Systems from Kyushu University, Fukuoka, Japan. He is presently an Assistant of Research Institute of Fundamental Information Science, Kyushu University. His research interests are algorithmic learning theory and its applications.



Takeshi Shinohara was born in Fukuoka, Japan in 1955. He received the B.S. degree in 1980 from Kyoto University, Kyoto, Japan, and the M.S. and Dr. Sci. degrees from Kyushu University, Fukuoka, Japan in 1982 and 1986, respectively. Currently, he is an Associate Professor of Department of Artificial Intelligence, Kyushu Institute of Technology, Fukuoka, Japan. His present interests include information retrieval, string pattern matching algorithms and algorithmic learning theory.



Akihiro Yamamoto was born in Kyoto, Japan in 1960. He received the B.S. degree from Kyoto University, Kyoto, Japan in 1985, and the M.S. degree and Dr. Sci. degrees from Kyushu University, Fukuoka, Japan in 1987 and 1990, respectively. Presently, he is a Lecturer of Department of Electrical Engineering, Hokkaido University, Hokkaido, Japan. His interests are in logic programming and unification algorithms.