

Learning Elementary Formal Systems and an Application to Discovering Motifs in Proteins

Miyano, Satoru

Research Institute of Fundamental Information Science Kyushu University

Shinohara, Ayumi

Research Institute of Fundamental Information Science Kyushu University

Shinohara, Takeshi

Department of Artificial Intelligence Kyushu Institute of Technology

<https://hdl.handle.net/2324/3144>

出版情報 : RIFIS Technical Report. 37, 1991-02-26. Research Institute of Fundamental
Information Science, Kyushu University

バージョン :

権利関係 :



RIFIS Technical Report

Learning Elementary Formal Systems and
an Application to Discovering Motifs in Proteins

Satoru Miyano
Ayumi Shinohara
Takeshi Shinohara

February 26, 1991
Revised: April 18, 1993

Research Institute of Fundamental Information Science
Kyushu University 33
Fukuoka 812, Japan

Learning Elementary Formal Systems and an Application to Discovering Motifs in Proteins

Satoru Miyano

miyano@rifis.sci.kyushu-u.ac.jp

Research Institute of Fundamental Information Science

Kyushu University 33, Fukuoka 812, Japan

Telephone: +81-92-641-1101 ex.4472

Ayumi Shinohara

ayumi@rifis.sci.kyushu-u.ac.jp

Research Institute of Fundamental Information Science

Kyushu University 33, Fukuoka 812, Japan

Takeshi Shinohara

shino@ai.kyutech.ac.jp

Department of Artificial Intelligence

Kyushu Institute of Technology, Iizuka 820, Japan

Abstract

An elementary formal system (EFS) is a logic program consisting of definite clauses whose arguments have patterns instead of first-order terms. We investigate EFSs for polynomial-time PAC-learnability and show by experiments on protein data that PAC-learning is very useful for discovering knowledge. A definite clause of an EFS is hereditary if every pattern in the body is a subword of a pattern in the head. With this new notion, we show that $\text{H-EFS}(m, k, t, r)$ is polynomial-time learnable, which is the class of languages definable by EFSs consisting of at most m hereditary definite clauses with predicate symbols of arity at most r , where k and t bound the number of variable occurrences in the head and the number of atoms in the body, respectively. The class defined by all finite unions of EFSs in $\text{H-EFS}(m, k, t, r)$ is also polynomial-time learnable. We also show an interesting series of **NC**-learnable classes of EFSs. As hardness results, the class of regular pattern languages is shown not polynomial-time learnable unless **RP=NP**. Furthermore, the related problem of deciding whether there is a common subsequence which is consistent with given positive and negative examples is shown **NP**-complete. As a practical application of our learning algorithm, we made experiments on learning transmembrane domains in proteins from amino acid sequences. The experimental results were quite successful.

key words

Elementary Formal System (EFS), polynomial-time PAC-learning, **NC**-learnable, pattern languages, common subsequence, structure prediction of proteins

Running head

Learning Elementary Formal Systems

1 Introduction

Genome Informatics is, roughly speaking, an area which aims at developing methods and tools for analyzing, understanding, and designing large molecules such as DNA and proteins with the aid of computers. Genome Informatics would be a challenging field for Machine Learning to show its identity and usefulness since it has been generating a lot of problems which should require machine learning technologies. One of the important issues in this field is to establish technologies for discovering knowledge from DNA and amino acid sequences that may provide new directions of investigations to biologists.

This paper deals with one of such problems, which is to find characteristic features of transmembrane domains of proteins (Hartmann et al., 1989) from amino acid sequences. Most approaches to this problem have been by means of biophysical analysis of amino acid residues (von Heijine, 1986) while our approach is based on concept learning from examples.

The applicability of concept learning largely depends on the representation of concepts. In this paper, as the representation of concepts, we use *elementary formal systems* (EFSs for short) which are a kind of logic programs introduced by (Smullyan, 1961) and applied to inductive inference (Gold, 1967) by (Arikawa et al., 1992b; Shinohara, 1990). Logic programs have been used for various knowledge representations in Genome Informatics. For instance, (Muggleton et al., 1992) gave an interesting approach to protein secondary structure prediction by inductive logic programming. On the other hand, “motifs” of functional domains are usually described with *patterns*, which are words containing variables. Such motifs have been compiled in the PROSITE database (Bairoch, 1991). Informally, an EFS is a logic program consisting of definite clauses whose arguments have patterns instead of first-order terms. Thus we can directly handle amino acid sequences with EFSs while keeping the structure of logic programs.

Machine Learning would make sound development by sound combination of

theory and practicisim. In this paper, we are involved with the model of PAC-learning (Valiant, 1984) that have provided many insightful theoretical results. As a theoretical investigation, we discuss which classes of elementary formal systems are polynomial-time learnable, while we also give some hardness results which imply the polynomial-time non-learnability of some classes. As practicisim, we consider an application of our learning algorithm developed in our theoretical study to identify transmembrane domains in proteins from amino acid sequences. There is a large gap between the feasibility in theory expressed as “polynomial time” and the feasibility in practice. However, by dealing with a class of EFSs of a much simpler form, our learning algorithm found very interesting hypotheses and the experiments for learning transmembrane domains are quite successful. These experimental results are given in Section 6. By the results in this paper and those in (Arikawa et al., 1992a; Shimozone et al., 1993), we are convinced that Machine Learning can be an important and strong strategy in Genome Informatics.

EFSs have a rich structure similar to logic programs and sound semantics (Yamamoto, 1992). Without any restriction, any recursively enumerable language can be defined by an EFS. In order to make learning from examples feasible, we shall restrict the form of definite clauses for EFSs. We introduce *hereditary* definite clauses that prove to be suited for learning from examples. We say that a definite clause is hereditary if every pattern appearing in the argument of the body is a subword of a pattern in the head. Since all patterns appearing in the body are directly inherited from those in the head, this property is helpful in finding a hypothesis from examples. We can also show that the languages defined by hereditary EFSs are in \mathbf{P} . For integers $m, k, t, r \geq 0$, we consider a hereditary EFS consisting of 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 at most t and the arity of each predicate symbol is at most r . Then $\text{H-EFS}(m, k, t, r)$ is defined as the class of languages definable by such EFSs. In Section 3, we show

that $\text{H-EFS}(m, k, t, r)$ is polynomial-time learnable for any $m, k, t, r \geq 1$ by showing that its dimension (Natarajan, 1989; Natarajan, 1991) is polynomial and it has a polynomial-time fitting (Natarajan, 1991). Moreover, we show that the class $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$ which is defined by taking all finite unions of concepts in $\text{H-EFS}(m, k, t, r)$ is also polynomial-time learnable by showing a polynomial-time Occam fitting for it. These results give an interesting series of polynomial-time learnable classes of EFS-languages. Some related topics on context-free grammars and ranked node rewriting grammars are discussed in (Abe, 1988).

In (Arikawa et al., 1992b), the *length-boundedness* of a definite clause is introduced, which requires that the length of the body is at most the length of the head for any substitution. We show that the concept class $\text{LB-H-EFS}(m, k, r)$, which is defined by length-bounded hereditary EFSs with at most m definite clauses such that the number of variable occurrences in the head is bounded by k and the arity of each predicate symbol is at most r , is **NC**-learnable, i.e., its learning algorithm can be parallelized efficiently. The notion of **NC**-learnable is due to (Vitter and Lin, 1992).

The class of pattern languages (Angluin, 1980) is exactly the same as $\text{LB-H-EFS}(1, *, *)$ except the empty set, where $*$ means “don’t care”. (Ko and Tzeng, 1991) showed that deciding whether there is a pattern consistent with given positive and negative examples is Σ_2^P -complete. Schapire (Schapire, 1990) strengthened this result in a sense by showing that the pattern languages cannot be learned in polynomial time *regardless of the representation* under a reasonable assumption. Hence even the pattern languages are hard to learn. In (Shinohara, 1982) a pattern of a simpler form called a regular pattern is considered, where each variable is allowed to occur in the pattern exactly once. Obviously the regular pattern languages are regular. In Section 5 we show that the consistency problem is **NP**-complete even for regular pattern languages. Therefore the class of regular pattern languages is not polynomial-time learnable if $\mathbf{RP} \neq \mathbf{NP}$. Moreover, we show the

related problem of deciding whether there is a subsequence which is common to given positive but not common to given negative examples is **NP**-complete. Similar results are obtained independently in (Jiang and Li, 1991). These negative results are not so strong as Schapire's result, but it is strong enough to convince of the hardness of its polynomial-time learnability. These **NP**-completeness results suggest that the number of occurrences of variables in a definite clause should be bounded if polynomial-time learnable subclasses of EFS languages are of interest.

2 Preliminaries

2.1 Patterns and elementary formal systems

Let Σ be a finite alphabet and $X = \{x_1, x_2, \dots\}$ be a set of variables. We assume that $\Sigma \cap X = \emptyset$. For an alphabet Δ , let Δ^* denote the set of all words over Δ , Δ^+ the set of all nonempty words, and $\Delta^{[n]}$ the set of all words of length n or less for $n \geq 0$.

A *pattern* is a word in $(\Sigma \cup X)^+$. A pattern π is called *regular* if each variable in π occurs exactly once in π . For instance, ax_1bx_2a is a regular pattern, but ax_1bx_1a is not, where a and b are in Σ . An *atom* is an expression of the form $p(\pi_1, \dots, \pi_r)$, where p is a predicate symbol with arity r and π_1, \dots, π_r are patterns. A *definite clause* is a clause of the form

$$A \leftarrow A_1, \dots, A_t,$$

where A, A_1, \dots, A_t are atoms and $t \geq 0$. The atom A is called the *head* and the part A_1, \dots, A_t the *body* of the definite clause. In case $t = 0$, we denote simply A instead of $A \leftarrow$. An *elementary formal system* (EFS for short) is a finite set of definite clauses.

A *substitution* θ is a homomorphism from patterns to patterns such that $\theta(a) = a$ for each $a \in \Sigma$. A substitution which maps some variables to the empty word is called an ε -*substitution*. In this paper, we do not allow any ε -substitutions without

extra notice. For a pattern π and a substitution θ , we denote by $\pi\theta$ the image of π by θ . For an atom $A = p(\pi_1, \dots, \pi_r)$ and a definite clause $C = A \leftarrow A_1, \dots, A_t$, we define $A\theta = p(\pi_1\theta, \dots, \pi_r\theta)$ and $C\theta = A\theta \leftarrow A_1\theta, \dots, A_t\theta$.

A definite clause C is *provable from* an EFS Γ , denoted by $\Gamma \vdash C$, if C is obtained from Γ by finitely many applications of substitutions and modus ponens. That is, the relation $\Gamma \vdash C$ is defined inductively as follows:

- (1) If $\Gamma \ni C$, then $\Gamma \vdash C$.
- (2) If $\Gamma \vdash C$, then $\Gamma \vdash C\theta$ for any substitution θ .
- (3) If $\Gamma \vdash A \leftarrow A_1, \dots, A_t, A_{t+1}$ and $\Gamma \vdash A_{t+1}$, then $\Gamma \vdash A \leftarrow A_1, \dots, A_t$.

For a predicate p with arity one, we define $L(\Gamma, p) = \{w \in \Sigma^+ \mid \Gamma \vdash p(w)\}$. A language $L \subseteq \Sigma^+$ is *definable by EFS* if there is an EFS Γ with a predicate symbol p with $L = L(\Gamma, p)$.

For a pattern π , the pattern language $L(\pi)$ is the set $\{w \in \Sigma^+ \mid w = \pi\theta \text{ for some substitution } \theta\}$ (Angluin, 1980). It should be noticed that a pattern language $L(\pi)$ is also defined by $L(\Gamma, p)$ with $\Gamma = \{p(\pi)\}$.

Example 1. Consider the following EFS with $\Sigma = \{a, b\}$:

$$\Gamma = \left\{ \begin{array}{l} p(x_1x_2) \leftarrow q(x_1), r(x_2) \\ q(ax_1b) \leftarrow q(x_1) \\ q(ab) \\ r(x_1x_1) \end{array} \right\}.$$

The language defined by Γ is $L(\Gamma, p) = \{a^n b^n w w \mid n \geq 1, w \in \{a, b\}^+\}$. In the definite clause $p(x_1x_2) \leftarrow q(x_1), r(x_2)$, the head is the atom $p(x_1x_2)$ and the atoms $q(x_1), r(x_2)$ form the body. A substitution is denoted as a collection of assignments $\{x_1 := \pi_1, \dots, x_n := \pi_n\}$. We can see $aabbaa \in L(\Gamma, p)$ as follows:

$$\begin{array}{ll}
C_1 = r(x_1x_1) & (\text{ axiom }) \\
C_2 = r(aa) & (C_1\{x_1 := a\}) \\
C_3 = q(ax_1b) \leftarrow q(x_1) & (\text{ axiom }) \\
C_4 = q(aabb) \leftarrow q(ab) & (C_3\{x_1 := ab\}) \\
C_5 = q(ab) & (\text{ axiom }) \\
C_6 = q(aabb) & (C_4 \& C_5) \\
C_7 = p(x_1x_2) \leftarrow q(x_1), r(x_2) & (\text{ axiom }) \\
C_8 = p(aabb\text{aa}) \leftarrow q(aabb), r(aa) & (C_7\{x_1 := aabb, x_2 := aa\}) \\
C_9 = p(aabb\text{aa}) \leftarrow q(aabb) & (C_2 \& C_8) \\
C_{10} = p(aabb\text{aa}) & (C_6 \& C_9)
\end{array}$$

Example 2. The languages $\{a^n b^n \mid n \geq 1\}$ and $\{a^n b^n c^n \mid n \geq 1\}$ are defined by the following EFSs Γ_1 and Γ_2 , respectively.

$$\Gamma_1 = \left\{ \begin{array}{l} p(ax_1b) \leftarrow p(x_1) \\ p(ab) \end{array} \right\}, \Gamma_2 = \left\{ \begin{array}{l} p(x_1x_2x_3) \leftarrow q(x_1, x_2, x_3) \\ q(ax_1, bx_2, cx_3) \leftarrow q(x_1, x_2, x_3) \\ q(a, b, c) \end{array} \right\}$$

2.2 Polynomial-time learnability

This section briefly reviews some necessary notions for PAC-learnability due to (Valiant, 1984).

We call a subset c of Σ^* a *concept*. A concept c can be regarded as a function $c : \Sigma^* \rightarrow \{0, 1\}$, where $c(w) = 1$ if w is in the concept and $c(w) = 0$ otherwise. A *concept class* is a nonempty set $\mathcal{C} \subseteq 2^{\Sigma^*}$ of concepts. We use a finite alphabet Λ for representing concepts. For a concept class \mathcal{C} , a *representation* is a function $R : \mathcal{C} \rightarrow 2^{\Lambda^*}$ such that $R(c)$ is a nonempty subset of Λ^* for c in \mathcal{C} and $R(c_1) \cap R(c_2) = \emptyset$ for any distinct concepts c_1 and c_2 in \mathcal{C} . For each $c \in \mathcal{C}$, $R(c)$ is the set of *names* for c . The length of a name $\nu \in R(c)$ is the word length $|\nu|$ of ν . We denote the length of the shortest name for c by $l_{min}(c, R)$. When R is clear from the context, we simply use $l_{min}(c)$.

An *example* is an element $\langle w, a \rangle$ in $\Sigma^* \times \{0, 1\}$. For a concept c , *example for a concept* c is a pair $\langle w, c(w) \rangle$ for $w \in \Sigma^*$. For a set $S \subseteq \Sigma^* \times \{0, 1\}$ of examples, we define $S_+ = \{w \mid \langle w, 1 \rangle \in S\}$ and $S_- = \{w \mid \langle w, 0 \rangle \in S\}$. We call a word in S_+ a *positive example* and a word in S_- a *negative example*, respectively. For two sets Y and N with $Y \cap N = \emptyset$, we say that a concept c is *consistent* with positive

examples in Y and negative examples in N if $c(w) = 1$ for all $w \in Y$ and $c(w') = 0$ for all $w' \in N$. A concept $c \in \mathcal{C}$ is *consistent* with a set S of examples if c is consistent with positive examples in S_+ and negative examples in S_- . For a set S of examples, $l_{\min}(S, R)$ is the length of the shortest name in R of any concept in \mathcal{C} which is consistent with S .

Definition 1. A concept class \mathcal{C} is *polynomial-time learnable* in a representation R if there exist an algorithm \mathcal{A} and a polynomial $\text{poly}(\cdot, \cdot, \cdot, \cdot)$ which satisfy the following conditions for any concept c in \mathcal{C} , any probability distribution P on $\Sigma^{[n]}$, and for any real numbers ε, δ ($0 < \varepsilon, \delta < 1$), and any integers $n \geq 0, s \geq 1$:

- (a) \mathcal{A} takes ε, δ, n , and s . The real numbers ε and δ are called the *accuracy* and *confidence*, respectively. The integers n and s are called the *length parameter* and the *concept complexity*, respectively.
- (b) \mathcal{A} may call EXAMPLE, which generates examples for the concept $c \in \mathcal{C}$, randomly according to the probability distribution P on $\Sigma^{[n]}$.
- (c) \mathcal{A} outputs a name $\nu \in R(h)$ for some concept $h \in \mathcal{C}$ satisfying $P(c \cup h - c \cap h) < \varepsilon$ with probability at least $1 - \delta$, when $l_{\min}(c) \leq s$ is satisfied.
- (d) The running time of \mathcal{A} is bounded by $\text{poly}(\frac{1}{\varepsilon}, \frac{1}{\delta}, n, s)$.

Remark 1. In (Natarajan, 1991), the input to the learning algorithm does not include the parameter s . This yields slightly different learnability. See (Haussler et al., 1988) for the equivalence and difference among these definitions of learnabilities.

Definition 2. (Blumer et al., 1989) Let \mathcal{C} be a concept class. We say that \mathcal{C} *shatters* a set $S \subseteq \Sigma^*$ if the set $\{c \cap S \mid c \in \mathcal{C}\}$ coincides with the set of *all* subsets of S . The *Vapnik-Chervonenkis dimension* of \mathcal{C} , denoted by $D_{VC}\mathcal{C}$, is the greatest

integer d such that there exists a set of cardinality d that is shattered by \mathcal{C} . For an integer $n \geq 0$, we define $\mathcal{C}^{[n]} = \{c \cap \Sigma^{[n]} \mid c \in \mathcal{C}\}$. We say that \mathcal{C} is of *polynomial dimension* if there exists a polynomial $d(n)$ such that $D_{VC}\mathcal{C}^{[n]} \leq d(n)$ for all $n \geq 0$.

Definition 3. A representation R for a concept class \mathcal{C} is *polynomial-time computable* if there exist a deterministic algorithm B and a polynomial q satisfying (a) and (b):

- (a) B takes as input a pair (w, ν) of words $w \in \Sigma^*$ and $\nu \in \Lambda^*$.
- (b) If $\nu \in R(c)$ for some $c \in \mathcal{C}$, then B halts in time $q(|w| + |\nu|)$ and outputs $c(w)$.

Definition 4. (Natarajan, 1991) Let \mathcal{C} be a concept class with representation R , and $S \subseteq \Sigma^* \times \{0, 1\}$ be a finite set of examples. A deterministic algorithm is said to be a *fitting* for \mathcal{C} in R if it takes as input S and outputs a name $\nu \in R$ of a concept $c \in \mathcal{C}$ which is consistent with S if any. A fitting is said to be a *polynomial-time fitting* if it runs in time polynomial in the length of its input and $l_{min}(S, R)$. A *randomized fitting* for \mathcal{C} in R is a randomized algorithm which takes as input S and outputs a name $\nu \in R$ of a concept $c \in \mathcal{C}$ which is consistent with S , if any, with probability greater than $\frac{1}{2}$. A fitting is an *Occam fitting* if there exist a polynomial q and a real number $0 \leq \alpha < 1$ such that for every input S , the output is of length at most $q(n, l_{min}(S, R))|S|^\alpha$, where $|S|$ is the number of examples in S and $n = \max\{|w| \mid \langle w, a \rangle \in S\}$.

The polynomial-time learnability is characterized as follows:

Lemma 1. (Haussler et al., 1988; Natarajan, 1989; Blumer et al., 1989; Natarajan, 1991) Let \mathcal{C} be a concept class and R be a polynomial-time computable representation for \mathcal{C} .

- (1) \mathcal{C} is polynomial-time learnable in R if \mathcal{C} is of polynomial dimension and there exists a polynomial-time fitting for \mathcal{C} in R .
- (2) \mathcal{C} is polynomial-time learnable in R if there exists a polynomial-time Occam fitting for \mathcal{C} in R .
- (3) \mathcal{C} is polynomial-time learnable in R only if there exists a randomized polynomial-time fitting for \mathcal{C} in R .

3 Polynomial-time learnable classes of EFSs

Consider a definite clause

$$q_0(\pi_1^0, \dots, \pi_{r_0}^0) \leftarrow q_1(\pi_1^1, \dots, \pi_{r_1}^1), q_2(\pi_1^2, \dots, \pi_{r_2}^2), \dots, q_t(\pi_1^t, \dots, \pi_{r_t}^t).$$

An EFS is defined as a finite collection of such clauses. In order to get polynomial-time learnable classes of EFSs, we focus our attention on the following features of an EFS.

- (1) The relationship between patterns in the head and patterns in the body.
- (2) The number of variables occurring in a pattern.
- (3) The number of atoms in the body.
- (4) The arity of a predicate.
- (5) The number of clauses in the EFS.

It has been shown in (Arikawa et al., 1992b) that any recursively enumerable set can be defined by an EFS whose clauses do not contain any internal variables, i.e., in each clause all variables in the body also appear in the head. By strengthening this view on the relationship between patterns in the head and patterns in the body, we define the following notion that is the key to finding polynomial-time learnable classes.

Definition 5. We say that a definite clause is *hereditary* if each pattern in the body is a subword of some pattern in the head. An EFS Γ is *hereditary* if each definite clause in Γ is hereditary.

Example 3. The definite clauses $p(ax_1bc) \leftarrow q(ax_1), r(x_1b)$ and $p(ax_1, bx_2, cx_3) \leftarrow q(x_1, x_2, x_3)$ are hereditary. But the definite clause $p(ax_1) \leftarrow q(bx_1)$ is not hereditary.

A pair (Γ, p) of an EFS and a predicate symbol is said to be *reduced* with respect to a nonempty set Y of words if $Y \subseteq L(\Gamma, p)$ and $Y \not\subseteq L(\Gamma', p)$ for any $\Gamma' \subsetneq \Gamma$.

As we shall show in Claim 3 of Lemma 5, languages definable by hereditary EFSs are in **P**. Moreover, hereditary EFSs have the following property that is very suited for learning from examples. The proof is obvious from the definition.

Lemma 2. Let Y be a nonempty set of words in Σ^+ and let (Γ, p) be a pair of a hereditary EFS and its predicate symbol of arity one. If (Γ, p) is reduced with respect to Y , then for each definite clause

$$q_0(\pi_1^0, \dots, \pi_{r_0}^0) \leftarrow q_1(\pi_1^1, \dots, \pi_{r_1}^1), q_2(\pi_1^2, \dots, \pi_{r_2}^2), \dots, q_t(\pi_1^t, \dots, \pi_{r_t}^t)$$

in Γ , there exists a substitution θ such that all $\pi_i^j \theta$'s are subwords of some $w \in Y$.

In Section 5, we shall show that we cannot get any polynomial-time learnable subclass of EFSs unless the number of variables occurring in a pattern is bounded by a constant. Hence it is reasonable to bound by a constant, say k , the number of variable occurrences in patterns for our purpose.

We do not have any strong mathematical reasons to restrict other three parameters; the number t of the atoms in the body, the arity r of a predicate, and the number m of the clauses. However, counting these parameters into consideration, we define the following class of EFS languages.

Definition 6. For $m, k, t, r \geq 0$, $\text{H-EFS}(m, k, t, r)$ is the class of languages definable by hereditary EFSs with at most m definite clauses each of which satisfies the following conditions:

- (a) The number of variable occurrences in the head is at most k .
- (b) The number of atoms in the body is at most t .
- (c) The arity of each predicate symbol is at most r .

The following two theorems give a series of classes of EFS languages that are polynomial-time learnable.

Theorem 1. $\text{H-EFS}(m, k, t, r)$ is polynomial-time learnable for any $m, k, t, r \geq 0$.

For a concept class \mathcal{C} , (Blumer et al., 1989) discussed the polynomial-time learnability of the *finite union class* of \mathcal{C} that is defined as

$$\mathcal{FU}(\mathcal{C}) = \{c_1 \cup c_2 \cup \dots \cup c_m \mid c_i \in \mathcal{C}, m \geq 1\}.$$

Though the concept class $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$ is not of polynomial dimension, we can prove the following theorem by showing a polynomial-time Occam fitting for this class. The idea of proof is mostly due to (Blumer et al., 1989). However, we shall give the details since for the application given in Section 6 its learning algorithm will play a key role in learning transmembrane domains of proteins.

Theorem 2. $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$ is polynomial-time learnable for any $m, k, t, r \geq 0$.

The rest of this section is devoted to the proofs of Theorem 1 and Theorem 2.

From Claim 3 of Lemma 5, the representation for the concept class $\text{H-EFS}(m, k, t, r)$ is polynomial-time computable if we use a conventional representation for EFSs.

We can also see that the representation for $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$ is polynomial-time computable.

First, we shall show Theorem 1 by proving that the conditions of Lemma 1 (1) are satisfied for $\text{H-EFS}(m, k, t, r)$ (Lemma 4 and Lemma 5).

We use the the following lemma:

Lemma 3. (Natarajan, 1989) A concept class \mathcal{C} is of polynomial dimension if and only if there exists a polynomial $p(n)$ such that $\log_2 |\mathcal{C}^{[n]}| \leq p(n)$ for all $n \geq 0$.

Let $\text{H-EFS}(m, *, t, r) = \bigcup_{k \geq 1} \text{H-EFS}(m, k, t, r)$ for $m, r, t \geq 0$. $\text{H-EFS}(m, *, t, r)$ is the class of languages defined by hereditary EFSs that allow an arbitrary number of variable occurrences in patterns in their definite clauses while other restrictions are kept as they are.

Lemma 4. $\text{H-EFS}(m, *, t, r)$ is of polynomial dimension for any $m, t, r \geq 0$.

Proof. Let $\text{H-EFS}(m, *, t, r)^{[n]} = \{L \cap \Sigma^{[n]} \mid L \in \text{H-EFS}(m, *, t, r)\}$ for $n \geq 0$. We evaluate the cardinality of $\text{H-EFS}(m, *, t, r)^{[n]}$. Let (Γ, p) be a pair of a hereditary EFS Γ and its predicate symbol p of arity one. Since the number of definite clauses is bounded by m , we need to consider only m predicate symbols, each of whose arity is at most r . Let $C = q_0(\pi_1^0, \dots, \pi_{r_0}^0) \leftarrow q_1(\pi_1^1, \dots, \pi_{r_1}^1), q_2(\pi_1^2, \dots, \pi_{r_2}^2), \dots, q_t(\pi_1^t, \dots, \pi_{r_t}^t)$ be a definite clause in Γ . Note that if $|\pi_i^0| > n$ for some pattern π_i^0 in the head, then $L(\Gamma, p) \cap \Sigma^{[n]} = L(\Gamma - \{C\}, p) \cap \Sigma^{[n]}$ since Γ is hereditary. Therefore we have only to consider definite clauses whose heads contain patterns of length at most n . Since the arity of the predicate symbol of the head is bounded by r , we see that the number of possible heads is at most $m(|\Sigma| + nr)^{nr}$.

Since we are dealing with a hereditary EFS, each pattern in the body of a definite clause is a subword of some pattern in the head. For each pattern in the body, the possible number of subwords from the head is at most $r \frac{n(n-1)}{2}$. Moreover, since the number of atoms in the body is at most t , the number of possible bodies

with the head is $(m(r^{\frac{n(n-1)}{2}})^r)^t$. Thus, the possible number of definite clauses is roughly bounded by $m(|\Sigma| + nr)^{nr} (m(r^{\frac{n(n-1)}{2}})^r)^t$.

Hence, the number of such hereditary EFSs is at most $(m(|\Sigma| + nr)^{nr} (m(r^{\frac{n(n-1)}{2}})^r)^t)^m$. Thus, we see that $\log_2 |\text{H-EFS}(m, *, t, r)^{[n]}|$ is $O(n \log n)$. By Lemma 3, $\text{H-EFS}(m, k, t, r)$ is of polynomial dimension. \square

It should be noticed that if the number of definite clauses is not bounded by a constant m then the resulting concept class is not of polynomial dimension since all finite sets can be defined.

Lemma 5. There exists a polynomial-time fitting for $\text{H-EFS}(m, k, t, r)$ for any $m, k, t, r \geq 0$.

Proof. Let S be a finite set of examples for some concept c in $\text{H-EFS}(m, k, t, r)$. If S_+ is empty, we choose a word $w \in \Sigma^+$ not in S_- and take a hereditary EFS $\Gamma = \{p(w)\}$. Then obviously $L(\Gamma, p)$ is consistent with S . Therefore, we assume that S_+ is not empty.

Let $\mathcal{G}(m, k, t, r, S_+)$ be the set of pairs (Γ, p) which satisfies the following conditions (1)-(3):

- (1) Γ contains at most m hereditary definite clauses such that in each clause the head has at most k variable occurrences, the body has at most t atoms, and each predicate is arity at most r .
- (2) For each patterns π in each definite clause in Γ , there is a substitution θ such that $\pi\theta$ is a subword of some positive example in S_+ .
- (3) Variables are from $\{x_1, \dots, x_k\}$ and predicate symbols are from $\{p, p_1, \dots, p_{m-1}\}$. The arity of p is fixed to be one, but we do not fix in advance the arity of p_i for $i = 1, \dots, m-1$. Hence the arity of p_i may differ from one EFS to another.

Claim 1. There exists a pair $(\Gamma, p) \in \mathcal{G}(m, k, t, r, S_+)$ such that the concept $L(\Gamma, p)$ is consistent with S .

Proof. Since S is the set of examples for some concept c in $\text{H-EFS}(m, k, t, r)$, there exists a pair (Γ_0, p) with $L(\Gamma_0, p) = c$ which satisfies the condition (1). Without loss of generality, we can assume that Γ_0 is reduced with respect to S_+ . Then Γ_0 satisfies the condition (2) by Lemma 2. Since Γ_0 is reduced and has at most m clauses, there are at most m distinct predicate symbols in Γ_0 . Therefore we can rename the variables and predicate symbols in Γ_0 so that they satisfy the condition (3).

Claim 2. $|\mathcal{G}(m, k, t, r, S_+)|$ is bounded by some polynomial in $\sum_{w \in S_+} |w|$.

Proof. Let $\Pi(k, S_+)$ be the set of patterns π such that π contains at most k variable occurrences which are from $\{x_1, \dots, x_k\}$, and $\pi\theta$ is a subword of some $w \in S_+$ for some substitution θ . Then $|\Pi(k, S_+)| \leq \sum_{w \in S_+} ((|w|^2)^{k+1} k!)$. The number of definite clauses with at most $t+1$ atoms, in which all patterns are from $\Pi(k, S_+)$ and all predicate symbols are from $\{p, p_1, \dots, p_{m-1}\}$ with arity at most r , is bounded by $(m|\Pi(k, S_+)|^r)^{t+1}$. Thus we have $|\mathcal{G}(m, k, t, r, S_+)| \leq ((m|\Pi(k, S_+)|^r)^{t+1})^m$. Since m, k, t , and r are constants, $|\mathcal{G}(m, k, t, r, S_+)|$ is bounded by a polynomial with respect to $\sum_{w \in S_+} |w|$.

Claim 3. There is an algorithm that, given a word w in Σ^+ and a hereditary EFS Γ satisfying (1), decides whether w is in $L(\Gamma, p)$ in polynomial time with respect to $|w| + |\Gamma|$, where $|\Gamma|$ represents the length of Γ as an expression.

Proof. We apply a bottom-up algorithm for deciding whether a given word w is in $L(\Gamma, p)$. Since Γ is hereditary, we need to consider only substitutions that substitute subwords of w to variables. Then, by such substitutions, we construct the family $\Gamma(w)$ of all clauses containing no variables that can be obtained from the clauses in Γ by replacing all variables by nonempty subwords of w . Since the number of variable occurrences, the arity of predicate symbols, and the number of atoms in the bodies are bounded by k, r , and t , respectively, and since the number

of definite clauses in Γ is also bounded by m , $\Gamma(w)$ contains at most polynomially many clauses. In order to check whether w is in $L(\Gamma, p)$, the bottom-up algorithm repeats applications of modus ponens to $\Gamma(w)$ until $p(w)$ is derived. Since $\Gamma(w)$ contains polynomially many clauses, we see that the algorithm runs in polynomial time with respect to $|w| + |\Gamma|$.

The polynomial-time algorithm finding the required hereditary EFS runs as follows: The algorithm enumerates pairs (Γ, p) in $\mathcal{G}(m, k, t, r, S_+)$. Then it checks by using the polynomial-time algorithm of Claim 3 whether $w \in L(\Gamma, p)$ for $w \in S_+$ and $w' \notin L(\Gamma, p)$ for $w' \in S_-$. If such pair is found, the algorithm outputs it as a hypothesis. \square

In order to prove Theorem 2, we show that there is a polynomial-time Occam fitting for $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$. Then Lemma 1 (2) yields the polynomial-time learnability.

We use the *weighted set cover problem* and its approximation algorithm *GreedyWSC* due to (Chvatal, 1979). The weighted set cover problem is, given a collection of finite sets T_1, \dots, T_n with positive real weights W_1, \dots, W_n , to find $J^* \subseteq I = \{1, \dots, n\}$ with $\bigcup_{i \in J^*} T_i = \bigcup_{i \in I} T_i$ such that $\text{weight}(J^*) = \sum_{i \in J^*} W_i$ is minimized.

The algorithm *GreedyWSC* is described in Figure 1:

Lemma 6. (Chvatal, 1979) For the weighted set cover problem, algorithm *GreedyWSC* runs in polynomial-time and produces a set cover $J \subseteq I$ with $\text{weight}(J) \leq \text{weight}(J^*) \cdot \log |I|$, where J^* is a minimal weighted set cover.

Proof. The basic idea is due to (Blumer et al., 1989). By Lemma 1 (2), we have only to show a polynomial-time Occam fitting for $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$. For a set S of examples, the algorithm *Occam* in Figure 2 finds in polynomial time a hypothesis which is consistent with S .

```

procedure GreedyWSC (  $\{\langle i, T_i, W_i \rangle\}_{i \in I}$  : set of triples ) : subset of  $I$ 
begin
   $UnCover := \bigcup_{i \in I} T_i$  ;
   $J := \emptyset$  ;
  while  $UnCover \neq \emptyset$ 
  begin
    Find  $k \in I$  which minimizes the ratio  $W_k/|T_k|$  ;
     $J := J \cup \{k\}$  ;
     $UnCover := UnCover - T_k$  ;
    foreach  $i \in I$ 
       $T_i := T_i - T_k$ 
    end
  return  $J$ 
end

```

Figure 1: Algorithm GreedyWSC

```

procedure Occam (  $S$  : set of examples ) : pair of hereditary EFS and predicate
begin
   $\mathcal{J} := \emptyset$ ; /* instance for the weighted set cover problem */
  foreach  $(\Gamma, p) \in \mathcal{G}(m, k, t, r, S_+)$ 
    if  $L(\Gamma, p) \cap S_- = \emptyset$  then
       $\mathcal{J} := \mathcal{J} \cup \{\langle \Gamma, L(\Gamma, p) \cap S_+, size(\Gamma) \rangle\}$ 
   $\mathcal{G} := GreedyWSC(\mathcal{J})$  ;
   $\Gamma^U = \bigcup_{\Gamma \in \mathcal{G}} \Gamma$  ;
  return  $(\Gamma^U, p)$ 
end

```

Figure 2: Occam fitting for $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$

Recall that we can generate all pairs in $\mathcal{G}(m, k, t, r, S_+)$ in polynomial-time with respect to $\sum_{w \in S_+} |w|$. Moreover, by Lemma 6, the size of the output Γ is at most as $\log |S|$ times as the size of minimum hereditary EFS which is consistent with S . Therefore the algorithm *Occam* is a polynomial-time Occam fitting for $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$. \square

4 NC-learnability

The notion of NC-learnability is introduced in (Vitter and Lin, 1992). By using NC algorithms instead of polynomial time algorithms, we can develop the same argument as that in Section 2 and obtain a similar result for NC-learnability.

The purpose of this section is to show a series of NC^2 -learnable subclasses of hereditary EFSs. It is shown in (Arimura, 1993) that a P-complete set can be described with a hereditary EFS. Therefore, at least, we cannot expect any NC-computable representation for $\text{H-EFS}(m, k, t, r)$. Hence we are required to get into a new class of EFSs.

Let $|\pi|$ denote the length of a pattern π . For an atom $p(\pi_1, \dots, \pi_n)$, we define $\|p(\pi_1, \dots, \pi_n)\| = |\pi_1| + \dots + |\pi_n|$. A definite clause $A \leftarrow A_1, \dots, A_t$ is called *length-bounded* if $\|A\theta\| \geq \|A_1\theta\| + \dots + \|A_t\theta\|$ for any substitution θ (Arikawa et al., 1992b). An EFS Γ is *length-bounded* if all definite clauses in Γ are length-bounded. For a length-bounded definite clause, we can easily see that, for each variable x_i in the clause, the number of occurrences of x_i in the head is not less than that in the body.

Example 4. The definite clause $p(ax_1) \leftarrow q(bx_1)$ is length-bounded but not hereditary, while $p(ax_1bc) \leftarrow q(ax_1), r(x_1b)$ is not length-bounded but hereditary. The definite clause $p(ax_1, bx_2, cx_3) \leftarrow q(x_1, x_2, x_3)$ is both length-bounded and hereditary.

Definition 7. For $m, k, r \geq 0$, $\text{LB-H-EFS}(m, k, r)$ is the class of languages definable by length-bounded hereditary EFSs 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 arity of each predicate is at most r .

Obviously the class $\text{LB-H-EFS}(m, k, r)$ contains infinitely many languages for any $m, k, r \geq 1$. Any context-free language is in $\text{LB-H-EFS}(m, 2, 1)$ for some $m \geq 1$ and any regular language is in $\text{LB-H-EFS}(m, 1, 1)$ for some $m \geq 1$.

Example 5. Example 2 shows that the language $\{a^n b^n \mid n \geq 1\}$ is in $\text{LB-H-EFS}(2, 1, 1)$ and that the language $\{a^n b^n c^n \mid n \geq 1\}$ is in $\text{LB-H-EFS}(3, 3, 3)$.

Remark 2. Unlike the definition of $\text{H-EFS}(m, k, t, r)$, we do not bound the number t of atoms in the body explicitly, when we define the class $\text{LB-H-EFS}(m, k, r)$. However, we can assume that the number of atoms in the body is at most k without loss of generality: Let (Γ, p) be a pair of a length-bounded EFS Γ and its predicate symbol of arity one. Assume that the number of variable occurrences in the head of each clause is at most k . For a definite clause $C = A \leftarrow A_1, \dots, A_t$ in Γ , suppose that an atom A_i does not contain any variables. If A_i is not provable from $\Gamma - \{C\}$, it is not hard to show $\Gamma \not\vdash A_i$ by induction on the number of applications of modus ponens since only modus ponens rules can eliminate A_i from the body of C . Hence the clause C is redundant in Γ , i.e., $L(\Gamma, p) = L(\Gamma - \{C\}, p)$. On the other hand, if A_i is provable from $\Gamma - \{C\}$, then the atom A_i is redundant in the clause C , i.e., $L(\Gamma, p) = L((\Gamma - \{C\}) \cup \{C'\}, p)$, where $C' = A \leftarrow A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_t$. Therefore we can assume that each atom contains at least one variable. From the length-boundedness of Γ , the total number of variable occurrences in the body is bounded by k . Thus the number of atoms in the body is at most k .

Theorem 3. $\text{LB-H-EFS}(m, k, r)$ is NC^2 -learnable for any $m, k, r \geq 0$.

Proof. We first observe that Claim 3 in the proof of Lemma 5 is solvable in \mathbf{NC}^2 for LB-H-EFS(m, k, r). Consider the following nondeterministic recursive procedure that returns *true* if and only if $\Gamma \vdash q(u_1, \dots, u_l)$ for a length-bounded hereditary EFS Γ and $u_1, \dots, u_l \in \Sigma^+$.

```

procedure Prove( $q, [u_1, \dots, u_l]$ )
begin
  guess a definite clause  $q(\pi_1^0, \dots, \pi_n^0) \leftarrow q_1(\pi_1^1, \dots, \pi_{r_1}^1), \dots, q_t(\pi_1^t, \dots, \pi_{r_t}^t)$  in  $\Gamma$ ;
  guess a substitution  $\theta$  with  $\pi_i^0 \theta = u_i$  ( $1 \leq i \leq l$ );
  if  $t = 0$  then return true
  elseif Prove( $q_j, [\pi_1^j \theta, \dots, \pi_{r_j}^j \theta]$ ) = true for all  $j = 1, \dots, t$  then return true
end

```

Figure 3: Algorithm for proving $\Gamma \vdash q(u_1, \dots, u_l)$

We can simulate the procedure *Prove* by a two-way nondeterministic auxiliary pushdown automaton that runs in polynomial time using $O(\log n)$ worktape space (Sudborough, 1978). We just state the idea of simulation. We assume that a word w and a pair (Γ, p) are given on the input tape. Since Γ is hereditary, we can assume that $x_h \theta$ is a subword of w for any variable x_h in the guessed clause. Therefore each $x_h \theta$ can be expressed by specifying the start and end positions of $x_h \theta$ in w . This requires only $O(\log n)$ worktape space. Moreover, since π contains at most k variables, we need $O(\log n)$ space to keep θ . Since Γ is on the input tape, guessing nondeterministically a definite clause from Γ requires $O(\log n)$ worktape space. Checking $\pi_i \theta = u_i$ is also possible in $O(\log n)$ worktape space. Recursions are simulated by a pushdown store in a conventional way by pushing $(q_j, [\pi_1^j \theta, \dots, \pi_{r_j}^j \theta])$ for $j = 1, \dots, t$, where each $\pi_i^j \theta$ is represented by a pair of binary integers in $O(\log n)$ bits. Since r_j is at most r , $(q_j, [\pi_1^j \theta, \dots, \pi_{r_j}^j \theta])$ requires $O(\log n)$ space. Then the simulation can be continued using $O(\log n)$ worktape space.

We consider the size of the recursion tree for *Prove*($p, [w]$), where each node is labeled with some *Prove*($q, [u_1, \dots, u_l]$). We can assume that each u_i of *Prove*($q, [u_1, \dots, u_l]$) is a subword of w . Since the number of predicates is at most m and the arity of

each predicate symbol is bounded by r , the depth of the recursion tree is bounded by some polynomial in $|w|$. Moreover, since Γ is length-bounded, the number of the leaves of this tree is at most $|w|$. Therefore, the number of the nodes in this recursion tree is also bounded by a polynomial in $|w|$. Hence, this nondeterministic algorithm accepts in polynomial time.

It is known in (Ruzzo, 1980; Ruzzo, 1981) that if a set is accepted by a polynomial time auxiliary pushdown automaton that uses $O(\log n)$ worktape space then it is in \mathbf{NC}^2 . Therefore it is possible to check in \mathbf{NC}^2 whether $\Gamma \vdash p(w)$. Thus we can decide the membership $w \in L(\Gamma, p)$ in \mathbf{NC}^2 when w and (Γ, p) are given as input. Therefore the representation for $\text{LB-H-EFS}(m, k, r)$ is computable in \mathbf{NC}^2 .

We combine this \mathbf{NC} algorithm for deciding membership with the following \mathbf{NC} algorithm. We consider how to generate all candidates (Γ, p) consistent with S_+ of positive examples and S_- of negative examples. We define $\mathcal{G}'(m, k, r, S_+)$ in the same way as $\mathcal{G}(m, k, t, r, S_+)$ in Lemma 5. From Lemma 2, it suffices to deal with $\mathcal{G}'(m, k, r, S_+)$ as the space of candidates. It is not hard to see that we can generate all pairs in $\mathcal{G}'(m, k, r, S_+)$ in \mathbf{NC}^2 since we are required to consider only subwords of the words in S_+ as patterns in atoms and m, k, r are constants.

From these \mathbf{NC}^2 algorithms, we can see that a pair (Γ, p) in $\mathcal{G}'(m, k, r, S_+)$ which is consistent with examples in S is computable in \mathbf{NC}^2 . \square

Theorem 4. $\mathcal{FU}(\text{LB-H-EFS}(m, k, r))$ is \mathbf{NC} learnable for any $m, k, r \geq 0$.

Proof. We can construct an \mathbf{NC} Occam fitting for $\mathcal{FU}(\text{LB-H-EFS}(m, k, r))$ by using the \mathbf{NC} approximate algorithm for the weighted set cover problem due to (Berger et al., 1989). \square

5 Regular patterns are hard to learn

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

Consistency Problem for \mathcal{C} in R

Instance : A set of examples $S \subseteq \Sigma^* \times \{0, 1\}$ with $S_+ \cap S_- = \emptyset$.

Question : Is there a name $\nu \in R$ of a concept $h \in \mathcal{C}$ which is consistent with S ?

If the consistency problem for \mathcal{C} in R is shown **NP**-complete, we can say that \mathcal{C} is not polynomial-time learnable in R under the assumption of **RP** \neq **NP**, since there is no randomized polynomial-time fitting for \mathcal{C} in R by Lemma 1 (3).

In this section, we deal with only regular patterns and abbreviate variables x_1, x_2, \dots by the same symbol x for simplicity.

Theorem 5. The consistency problem for the class of regular pattern languages is **NP**-complete.

Proof. Obviously the problem is in **NP**. We give a polynomial-time reduction from 3SAT to the problem. Let $F = C_1 \cdots C_m$ be a formula in 3-CNF with variables u_1, u_2, \dots, u_n . Without loss of generality, we can assume that C_i does not contain both u_k and $\overline{u_k}$ for any $1 \leq k \leq n$. We define Y of positive examples and N of negative examples so that F is satisfiable if and only if there is a regular pattern π consistent with Y and N .

First, we use $n + 1$ positive examples s_0, \dots, s_n and $n + 1$ negative examples t_0, \dots, t_n over $\Sigma = \{0, 1\}$ so that any consistent regular pattern π must be of the form $\pi = \tau_1 \tau_2 \cdots \tau_n$, where $\tau_i = 0x$ or $x0$ ($1 \leq i \leq n$). We define

$$s_0 = (00)^n;$$

$$s_i = (00)^{i-1} 010(00)^{n-i} \text{ for } 1 \leq i \leq n;$$

$$t_0 = 0^{2n-1};$$

$$t_i = (00)^{i-1} 11(00)^{n-i} \text{ for } 1 \leq i \leq n.$$

From the positive example s_0 with $|s_0| = 2n$ and the negative example t_0 with $|t_0| = 2n - 1$, we see that any pattern π consistent with s_0 and t_0 satisfies $\pi \in$

$\{0, x\}^+$ and $|\pi| = 2n$. Thus we can denote $\pi = \sigma_1\sigma_2\cdots\sigma_{2n}$ with $\sigma_k \in \{0, x\}$ for $1 \leq k \leq 2n$.

For $1 \leq i \leq n$, since $|s_i| = 2n+1$ and ε -substitutions are not allowed, the $(2i)$ th character 1 of s_i must match with either σ_{2i-1} or σ_{2i} of pattern π , i.e., either σ_{2i-1} or σ_{2i} is x . On the other hand, both of them are not x since $t_i \notin L(\pi)$. Therefore each $\tau_i = \sigma_{2i-1}\sigma_{2i}$ is $0x$ or $x0$ for $1 \leq i \leq n$.

For C_1, \dots, C_m of F , we use additional negative examples d_1, \dots, d_m to forbid that all of the three literals in C_i are assigned to *false* for $1 \leq i \leq m$. For $1 \leq i \leq m$, we define

$$d_i = r_1r_2\cdots r_n, \text{ where } r_k = \begin{cases} 01 & \text{if literal } u_k \text{ appears in } C_i \\ 10 & \text{if literal } \overline{u_k} \text{ appears in } C_i \\ 00 & \text{otherwise.} \end{cases}$$

Since we have assumed that both u_k and $\overline{u_k}$ does not appear in any C_i , the above d_i is well-defined. Then let $Y = \{s_0, s_1, \dots, s_n\}$ and $N = \{t_0, t_1, \dots, t_n, d_1, \dots, d_m\}$.

Assume that F is satisfiable under a truth assignment $\hat{u}_1, \dots, \hat{u}_n$. Then we define a regular pattern $\pi = \tau_1\cdots\tau_n$ by putting $\tau_i = x0$ if $\hat{u}_i = \textit{true}$ and $\tau_i = 0x$ if $\hat{u}_i = \textit{false}$ for $1 \leq i \leq n$. It is easy to see from the definitions of Y and N that π is consistent with Y and N . In fact, it is clear that $s_i \in L(\pi)$ and $t_i \notin L(\pi)$ for $0 \leq i \leq n$. Since F is satisfiable by the assumption, each C_i contains either u_k with $\hat{u}_k = \textit{true}$ or $\overline{u_k}$ with $\hat{u}_k = \textit{false}$ for some k . In the former case, $\tau_k = x0$ and $r_k = 01$ guarantee $d_i \notin L(\pi)$. In the latter case, $\tau_k = 0x$ and $r_k = 10$ give an evidence for $d_i \notin L(\pi)$ similarly.

Conversely, we assume that there exists a regular pattern π consistent with Y and N . Then π must be of the form $\tau_1\tau_2\cdots\tau_n$, where $\tau \in \{0x, x0\}$, since π is consistent with s_i 's and t_i 's for $0 \leq i \leq n$. We define the truth assignment $\hat{u}_i = \textit{true}$ if $\tau_i = x0$ and $\hat{u}_i = \textit{false}$ if $\tau_i = 0x$ for $1 \leq i \leq m$. For each C_i , at least one literal must be assigned to *true* since $L(\pi)$ does not contain the negative example d_i . \square

Example 6. For an instance $F = (u_1 + \overline{u_2} + u_3) \cdot (\overline{u_1} + u_3 + u_4)$ of 3SAT, we construct five positive examples $\{00000000, 01000000, 00010000, 00000100, 000000010\}$

and seven negative examples $\{0000000, 11000000, 00110000, 00001100, 00000011, 01100100, 10000101\}$. A regular pattern consistent with these examples, say $\pi = x0x00x0$, corresponds to the truth assignment $\hat{u}_1 = true, \hat{u}_2 = true, \hat{u}_3 = false, \hat{u}_4 = true$, which satisfies the formula F .

Theorem 6. Let $m \geq 1$ be an integer. The consistency problem for the class

$$\{L(\pi_1) \cup \dots \cup L(\pi_m) \mid \pi_i\text{'s are regular patterns}\}$$

is NP-complete for any $m \geq 1$.

Proof. The case of $m = 1$ is Theorem 5. For $m \geq 2$, we will reduce the problem to the case of $m = 1$. Let Y and N be the sets of positive examples and negative examples given in the proof of Theorem 5. Then we define as follows:

$$Y' = Y \cup \{1^i \mid 1 \leq i \leq m - 1\},$$

$$N' = N \cup \{1^m\}.$$

We will show that the following two statements are equivalent:

- (1) There is a set M consisting of m regular patterns such that $\bigcup_{\pi \in M} L(\pi)$ is consistent with Y' and N' .
- (2) There is a regular pattern π_0 such that $L(\pi_0)$ is consistent with Y and N .

We show that (1) implies (2). For each $i = 1, \dots, m - 1$, the set M contains a regular pattern π with $1^i \in L(\pi)$ since 1^i is a positive example. Then π is in $\{1, x\}^{[i]}$. However, if a variable occurs in π , the negative example 1^m is also in $L(\pi)$. Therefore, $\pi = 1^i$. Thus M contains $m - 1$ patterns $1, 11, \dots, 1^{m-1}$ without any variables. Therefore M must contain a regular pattern π_0 such that $L(\pi_0)$ is consistent with Y and N . The converse is almost clear. \square

An extended regular pattern language $\tilde{L}(\pi)$ (Shinohara, 1983) is defined by allowing ε -substitutions to a regular pattern π . The ε -substitutions might change the behavior entirely since the length of the possible patterns can not be bounded. However, any extended regular pattern language can be defined by a regular pattern in *canonical form* $\pi = w_0 x w_1 x \cdots x w_{n-1} x w_n$ with $w_0, w_n \in \Sigma^*$ and $w_i \in \Sigma^+$ for $1 \leq i \leq n-1$, since two consecutive variables are reduced to a single variable (Shinohara, 1983).

Theorem 7. Let $m \geq 1$ be an integer. The consistency problem for the class

$$\{\tilde{L}(\pi_1) \cup \cdots \cup \tilde{L}(\pi_m) \mid \pi_i \text{'s are regular patterns}\}$$

is NP-complete.

Proof. We will show only the case of $m = 1$. For $m \geq 2$, we can reduce the problem to the case of $m = 1$ in the same way as the proof of Theorem 6.

The basic idea of the proof is similar to that of Theorem 5. For an instance $F = C_1 \cdots C_m$ of 3SAT with variables u_1, u_2, \dots, u_n , we give the following two positive examples over $\Sigma = \{0, 1, \#\}$:

$$s_1 = 0\#0\#\cdots\#0 \text{ with } |s_1| = 2n - 1,$$

$$s_2 = 00\#00\#\cdots\#00 \text{ with } |s_2| = 3n - 1.$$

We also use $2n - 1$ negative examples \check{t}_i 's and t_i 's:

\check{t}_i : the word obtained by deleting the i th $\#$ of s_1 , $1 \leq i \leq n - 1$,

t_i : the word obtained by replacing the i th 0 of s_1 with 101, $1 \leq i \leq n$.

It can be noticed that a regular pattern in canonical form which is consistent with positive examples s_1, s_2 and negative examples $\check{t}_1, \dots, \check{t}_{n-1}, t_1, \dots, t_n$ must be of the form:

$\pi = \tau_1 \# \tau_2 \# \dots \# \tau_n$, where $\tau_i \in \{0x, x0\}$ for $1 \leq i \leq n$.

Then for each C_i of F , we define the following additional negative examples:

$$d_i = r_1 \# r_2 \# \dots \# r_n, \text{ where } r_k = \begin{cases} 01 & \text{if literal } u_k \text{ appears in } C_i \\ 10 & \text{if literal } \overline{u_k} \text{ appears in } C_i \\ 0 & \text{otherwise.} \end{cases}$$

We can verify that there is a regular pattern π such that $\tilde{L}(\pi)$ is consistent with these examples if and only if F is satisfiable in the same way. \square

The common subsequence problem has been dealt from independent viewpoints, such as text processing, data compression, or DNA sequences (Maier, 1978; Wagner and Fischer, 1974). It is known that the longest common subsequence problem is **NP**-complete (Maier, 1978). The problem of finding a sequence $a_1 \dots a_n$ which is common to all positive examples but not common to any of negative examples is equivalent to finding a regular pattern $\pi = xa_1x \dots xa_nx$ such that the extended regular pattern language $\tilde{L}(\pi)$ is consistent with the positive and negative examples. Thus we call a regular pattern of the form $\pi = xa_1xa_2x \dots xa_nx$ with $a_i \in \Sigma$ for $1 \leq i \leq n$ a *common subsequence*.

Theorem 8. The consistency problem for the class of common subsequence languages is **NP**-complete.

Proof. Since the basic idea is also similar, details are omitted. Two positive examples

$$s_1 = 01\#01\#\dots\#01 \text{ with } |s_1| = 3n - 1,$$

$$s_2 = 10\#10\#\dots\#10 \text{ with } |s_2| = 3n - 1$$

and $2n - 1$ negative examples

\check{t}_i : the word obtained by deleting the i th $\#$ of s_1 for $1 \leq i \leq n - 1$,

t_i : the word obtained by deleting the i th 01 of s_1 for $1 \leq i \leq n$

make a consistent common subsequence restricted to the form:

$$\pi = a_1 \# a_2 \# \dots \# a_n, \text{ where } a_i \in \{0, 1\} \text{ for } 1 \leq i \leq n.$$

Additional m negative examples

$$d_i = r_1 \# r_2 \# \dots \# r_n, \text{ where } r_k = \begin{cases} 0 & \text{if literal } u_k \text{ appears in } C_i \\ 1 & \text{if literal } \overline{u_k} \text{ appears in } C_i \\ 01 & \text{otherwise,} \end{cases}$$

guarantee the equivalence between the existence of a consistent common subsequence and the satisfiability of a given 3-CNF formula. \square

6 Application to learning from amino acid sequences

The primary structure of a protein is described as a sequence of amino acid residues of 20 kinds. One of the important problems in Genome Informatics is to discover rules for predicting functions of proteins by analyzing their amino acid sequences.

The purpose of this section is to apply our learning strategy for identifying transmembrane domains of proteins (Hartmann et al., 1989; von Heijine, 1988). We will show some experiments using real protein data with successful results.

6.1 Membrane proteins and PIR database

Figure 4 shows an example of an amino acid sequence of a membrane protein of a Norway rat. There is a tendency to assume that a membrane protein has transmembrane domains each of which constitutes an α -helix structure generating the membrane. The protein in Figure 4 has four transmembrane domains. The reported length of a transmembrane domain is not large, usually, $20 \sim 30$. If a sequence corresponding to a transmembrane domain is found in a protein, the probability that it is a membrane protein will get larger. Therefore it is important to identify transmembrane domains in amino acid sequences.

These amino acid sequences have been compiled in the PIR database (PIR, 1991) together with their additional information such as functions. This database

is growing larger year by year, and currently its size is about 50MB.

FEATURE

```

19-41          #Domain transmembrane
101-123       #Domain transmembrane
133-156       #Domain transmembrane
195-218       #Domain transmembrane

```

SEQUENCE

```

          5          10          15          20          25          30
1  M D V V N Q L V A G G Q F R V V K E P L G F V K V L Q W V F
31 A I F A F A T C G S Y T G E L R L S V E C A N K T E S A L N
61 I E V E F E Y P F R L H Q V Y F D A P S C V K G G T T K I F
91 L V G D Y S S S A E F F V T V A V F A F L Y S M G A L A T Y
121 I F L Q N K Y R E N N K G P M M D F L A T A V F A F M W L V
151 S S S A W A K G L S D V K M A T D P E N I I K E M P M C R Q
181 T G N T C K E L R D P V T S G L N T S V V F G F L N L V L W
211 V G N L W F V F K E T G W A A P F M R A P P G A P E K Q P A
241 P G D A Y G D A G Y G Q G P G G Y G P Q D S Y G P Q G G Y Q
271 P D Y G Q P A S G G G G Y G P Q G D Y G Q Q G Y G Q Q G A P
301 T S F S N Q M

```

Figure 4: An amino acid sequence of a membrane protein containing four transmembrane domains.

In applying our learning strategy to this problem, we regard the sequences of transmembrane domains as positive examples. The PIR database contains the amino acid sequences with FEATURE field where transmembrane domains are indicated.

For example, in Figure 4 the transmembrane domains of the amino acid sequence w are indicated by intervals 19-41, 101-123, 133-156, 195-218. Then the substrings $w[19..41]$, $w[101..123]$, $w[133..156]$, and $w[195..218]$ are taken as positive examples.

As negative examples, we use amino acid sequences without any overlap with transmembrane domains. Since the length of a positive example is $20 \sim 30$, we randomly choose sequences of length around 30 for negative examples.

We collect all the positive examples from the PIR database and the same num-

ber of negative examples as shown in Table 6.1. Since the PIR database is not completely correct, the data may contain some noises.

Sequences	Positive	Negative
Transmembrane	689	689

A hydropathy plot (Engelman et al., 1986; Kyte and Doolittle, 1982; Rao and Argos, 1986) has been used generally to predict transmembrane domains from primary sequences. Instead of dealing with twenty symbols of amino acids, we classify these symbols into three classes by the hydropathy indices of amino acids (Kyte and Doolittle, 1982). More precisely, we transform symbols by Table 1.

Amino Acids	Hydropathy	New Symbol
A M C F L V I	1.8 ~ 4.5	*
P Y W S T G	-1.6 ~ -0.4	+
R K D E N Q H	-4.5 ~ -3.2	-

Table 1: Transformation rules

This transformation from 20 symbols to 3 symbols reduces the search space of hypotheses. The sequence in Figure 5 is the result of this transformation from the sequence in Figure 4.

```

* - * * - - * * * + + - * - * * - -(+ * + * * - * * - + * *
* * * * * * + * + + +) + + - * - * + * - * * - - + - + * * -
* - * - * - + + * - * - - * + * - * + + * * - + + + + - * *
* * + - + + + + * -( * * * + * * * * * * * + + * + * * * + +
* * *) - - - + - - - - -(+ + * * - * * * + * * * * * * + * *
+ + + * + *) - + * + - * - * * + - + - - * * - - * + * * - -
+ + - + * - - * - - + * + + (+ * - + + * * * + * * - * * * +
* + - * + * * *) - - + + + * * + * * - * + + + * + - - - + *
+ + - * + + - * + + + - + + + + + + + - - + + + + - + + + -
+ - + + - + * + + + + + + + + - + - + + - - + + + - - + * +
+ + * + - - *

```

Figure 5: The sequence obtained by the transformation, where transmembrane domains are indicated by parentheses.

Positive examples	Negative examples
+++**+*****+*--*****-+	*+-----**-----*--+-***
+++*****-*****+*****+	-+++++-----**--+-*-*
+++*****+	*****+*--*+---++++-
+++*****+*****-*****-*	**---*****+---*---**---*
++*****+*****-***+	+**+*---*+*+---+*****+*--**+-
++-*****+*****+*****+---+	**+*-+*+---+*****+*****+
+*****+*****+*****+*****	-+*-+*+---+*****-**---*+*+*
*****+*****+*****-*****	-*---+---+*+---+*****+*****
+***+*****+*****	-+---**---+*+---+*+---*+*+*+---
*+*****-+*+*+*****	+---+*+*+---*+*+*+*+*+*+*+*+---

Figure 6: Positive and negative examples

As is seen, this transformation makes the characteristics of a transmembrane domain more vivid. Figure 6 gives some of the sequences obtained by this transformation from the transmembrane domains chosen from our 689 examples.

We denote by *POS* and *NEG* the sets of these positive and negative examples converted by Table 1, respectively. Fortunately, *POS* and *NEG* do not have any overlaps.

6.2 Experiments and results

As a hypothesis space, we use the concept class defined as

$$\{\tilde{L}(\pi_1) \cup \dots \cup \tilde{L}(\pi_m) \mid \pi_i \text{ is a regular pattern in } \Pi' \text{ and } m \geq 1\},$$

where Π' is the set of regular patterns of the following forms:

$$x_1 \alpha_1 x_2,$$

$$x_1 \alpha_1 x_2 \alpha_2 x_3,$$

$$x_1 \alpha_1 x_2 \alpha_2 x_3 \alpha_3 x_4$$

with $\alpha_1, \alpha_2, \alpha_3$ in $\{*, +, -\}^+$.

We can see from an argument similar to Theorem 2 that this concept class is polynomial-time learnable by the algorithm *Occam* in Section 3.

The reason why we restrict the number of variables to 4 is simply due to the actual time and space required to implement the algorithm.

Our learning algorithm chooses randomly two small sets *pos* and *neg* from *POS* and *NEG*, respectively. Then the sample *S* defined by *pos* and *neg* is given to an input to the algorithm *Occam*. This process shall be repeated until good hypotheses are found.

In our experiments, the size of *pos* (*neg*) varies from 5 to 20.

6.2.1 Finding patterns from positive training examples

The first approach we take for this problem is to find a collection of regular patterns which covers almost all positive training examples and excludes almost all negative training examples.

Table 2 shows the good hypotheses and their success rates that our learning system has produced.

Patterns	Positive	Negative
$x_1*****x_2$	72.6%	10.2%
accuracy	72.6%	89.8%

(P1)

Patterns	Positive	Negative
x_1++++x_2	38.6%	9.6%
$x_1***x_2**x_3$	80.4%	19.7%
accuracy	82.1%	76.2%

(P2)

Table 2: Collections of regular patterns covering positive examples and excluding negative examples that was produced by our learning algorithm from 10 positive (transmembrane domain) and 10 negative (nontransmembrane domain) training examples. The second (third, resp.) column shows the percentage of the positive (negative, resp.) examples that the pattern in the first column covers. The last row shows the accuracy for positive examples and negative examples.

6.2.2 Finding patterns from negative training examples

We are also interested in collections of regular patterns which *exclude* positive examples and *cover* negative examples. That is to say, we use transmembrane domains as negative examples, and nontransmembrane domains as positive examples.

Table 3 shows the results.

Pattern	Positive	Negative
x_1--x_2	7.4%	87.7%
accuracy	92.6%	87.7%

(N1)

Pattern	Positive	Negative
x_1--x_2	7.4%	87.7%
x_1++-x_2	7.0%	53.3%
accuracy	87.7%	95.1%

(N2)

Table 3: Collections of regular patterns covering nontransmembrane domains and excluding transmembrane domains. 10 positive and negative training examples are used.

As is seen, hypotheses (N1) and (N2) are very small and the success rates for both positive and negative examples are quite good. From these observations, we can say that the approach from negative examples is much better than that from positive examples in the last section.

After recognizing the importance of negative examples, we have finally found the following pattern:

$$x_1-x_2-x_3-x_4-x_5$$

Table 4 (N3) shows the result whose accuracy is more than 91%.

Patterns	Positive	Negative
$x_1-x_2-x_3-x_4-x_5$	8.4%	94.8%
accuracy	91.6%	94.8%

(N3)

Table 4: Results for $x_1-x_2-x_3-x_4-x_5$

7 Concluding remarks

We showed that the classes $\text{H-EFS}(m, k, t, r)$ and $\mathcal{FU}(\text{H-EFS}(m, k, t, r))$ are polynomial-time learnable for any constant $m, k, t, r \geq 0$. But if the number k of variable occurrences in the head of each definite clause is not bounded by a fixed constant, even for some small subclasses of $\text{H-EFS}(m, *, t, r)$, we can not expect polynomial-time learning. For example, the consistency problem for $\text{H-EFS}(1, *, 0, 1)$, which is the class of pattern languages, is known to be Σ_2^P -complete (Ko and Tzeng, 1991). We strengthened this observation by showing the **NP**-completeness of the consistency problem for the class defined by taking all unions of m regular pattern languages for any fixed constant $m \geq 1$. According to the number m of the clauses, we know that the class $\text{H-EFS}(*, k, t, r)$ is not of polynomial dimension. Note that, however, it does not necessarily imply that $\text{H-EFS}(*, k, t, r)$ is not polynomial-time learnable, since Occam fitting for $\text{H-EFS}(*, k, t, r)$ might exist. As to another parameters in $\text{H-EFS}(m, k, t, r)$, we do not have any results which justify the restriction on the number t of atoms in the body and the arity r of a predicate symbol, in order to meet polynomial-time learnability.

We introduced hereditary definite clauses for EFSs in order to make it feasible to learn a concept from given examples. However, the experiments in Section 6 did not have the benefit of hereditariness directly since we used the class of finite unions of regular pattern languages, which have no bodies. This is because the running time of our learning algorithm, which is essentially an enumeration method, is huge although it remains polynomial. In order to finish the computation in realistic

time, we had to restrict the class as in Section 6. Therefore it is an important problem to develop a more efficient learning algorithm which makes good use of the hereditariness.

In the experiments of learning EFSs, we met with quite satisfactory results on identification of transmembrane domains in amino acid sequences from positive and negative data. We have also taken another approach to this problem by learning decision trees over regular patterns (Arikawa et al., 1992a; Shimozono et al., 1993). In comparison, since the algorithm for finding decision trees is much more efficient than that in this paper, we met a better performance in time and accuracy in the experiments, while the algorithm in (Arikawa et al., 1992a; Shimozono et al., 1993) is not fully analyzed and the accuracy of outputs is not theoretically guaranteed.

Acknowledgments

We would like to thank Setsuo Arikawa, Hiroki Arimura, Satoru Kuhara and Shinichi Shimozono for discussions. The work is partially supported by Grant-in-Aid for Scientific Research on Priority Areas, "Genome Informatics" from the Ministry of Education, Science and Culture, Japan.

A part of this work was presented at the Second Workshop on Algorithmic Learning Theory (ALT'91), Tokyo, Japan, 1991.

References

- Abe, N. (1988). Polynomial learnability and locality of formal grammars. In *Proceedings of the Twenty Sixth Meeting of the Association for Computational Linguistics*.
- Angluin, D. (1980). Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62.

- Arikawa, S., Kuhara, S., Miyano, S., Mukouchi, Y., Shinohara, A., and Shinohara, T. (1992a). A machine discovery of a negative motif from amino acid sequences by decision trees over regular patterns. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, pages 618–625 (to appear in *New Generation Computing* 11, (1993)).
- Arikawa, S., Shinohara, T., and Yamamoto, A. (1992b). Learning elementary formal systems. *Theoretical Computer Science*, 95:97–113.
- Arimura, H. (1993). Personal communication.
- Bairoch, A. (1991). PROSITE: A dictionary of sites and patterns in proteins. *Nucleic Acids Research*, 19:2241–2245.
- Berger, B., Rompel, J., and Shor, P. (1989). Efficient NC algorithms for set cover with application to learning and geometry. In *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, pages 54–59.
- Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965.
- Chvatal, V. (1979). A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4(3):233–235.
- Engelman, D.M., Steiz, T.A., and Goldman, A. (1986). Identifying nonpolar transbilayer helices in amino acid sequences of membrane proteins. *Annual Review of Biophysics and Biophysical Chemistry*, 15:321–353.
- Gold, E. (1967). Language identification in the limit. *Information and Control*, 10:447–474.
- Hartmann, E., Rapoport, T.A., and Lodish, H.F. (1989). Predicting the orientation of eukaryotic membrane-spanning proteins. *Proceedings of the National Academy of Science of the United States of America*, 86:5786–5790.

- Haussler, D., Kearns, M., Littlestone, N., and Warmuth, M. (1988). Equivalence of models for polynomial learnability. In *Proceedings of the First Workshop on Computational Learning Theory*, pages 34–50.
- von Heijne, G. (1986). A new method for predicting signal sequence cleavage sites. *Nucleic Acids Research*, 14(11):4683–4690.
- von Heijne, G. (1988). Transcending the impenetrable: how proteins come to terms with membranes. *Biochimica et Biophysica Acta*, 947:307–333.
- Jiang, T. and Li, M. (1991). On the complexity of learning strings and sequences. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 367–371.
- Ko, K. and Tzeng, W. (1991). Three Σ_2^P -complete problems in computational learning theory. *Computational Complexity*, 1(3):269–310.
- Kyte, J. and Doolittle, R. (1982). A simple method for displaying the hydrophobic character of protein. *Journal of Molecular Biology*, 157:105–132.
- Maier, D. (1978). The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25:322–336.
- Muggleton, S., King, R., and Sternberg, M. (1992). Using logic for protein structure prediction. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, Vol. I*, pages 685–696.
- Natarajan, B. (1989). On learning sets and functions. *Machine Learning*, 4(1):67–97.
- Natarajan, B. (1991). *Machine Learning — A Theoretical Approach*. Morgan Kaufmann Publishers.
- PIR (1991). Protein identification resource. National Biomedical Research Foundation.

- Rao, J. and Argos, P. (1986). A conformational preference parameter to predict helices in integral membrane proteins. *Biochimica et Biophysica Acta*, 869:197–214.
- Ruzzo, W. (1980). Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21(2):218–235.
- Ruzzo, W. (1981). On uniform circuit complexity. *Journal of Computer and System Sciences*, 22(3):365–383.
- Schapire, R. (1990). Pattern languages are not learnable. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 122–129.
- Shimozono, S., Shinohara, A., Shinohara, T., Miyano, S., Kuhara, S., and Arikawa, S. (1993). Finding alphabet indexing for decision trees over regular patterns: an approach to bioinformatical knowledge acquisition. In *Proceedings of the Twenty-Sixth Hawaii International Conference on System Sciences, Vol. I*, pages 763–773.
- Shinohara, T. (1982). Polynomial time inference of pattern languages and its applications. In *Proceedings of the Seventh IBM Symposium on Mathematical Foundation of Computer Science*, pages 191–209.
- Shinohara, T. (1983). Polynomial time inference of extended regular pattern languages. *Lecture Notes in Computer Science*, 147:115–127.
- Shinohara, T. (1990). Inductive inference from positive data is powerful. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 97–110.
- Smullyan, R. (1961). *Theory of Formal Systems*. Princeton University Press.
- Sudborough, I. (1978). On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414.

- Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.
- Vitter, J. S. and Lin, J.-H. (1992). Learning in parallel. *Information and Computation*, 96:179–202.
- Wagner, R. and Fischer, M. (1974). The string-to-string correction problem. *Journal of the ACM*, 21:168–73.
- Yamamoto, A. (1992). Procedural semantics and negative information of elementary formal system. *Journal of Logic Programming*, 13(1):89–97.