Learning elementary formal systems*

Setsuo Arikawa

Research Institute of Fundamental Information Science, Kyushu University 33, Fukuoka 817, Japan

Takeshi Shinohara

Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka 820, Japan

Akihiro Yamamoto**

Department of Information Systems, Kyushu University 39, Kasuga 816, Japan

Communicated by M. Nivat
Received October 1989

Abstract


The elementary formal systems (EFS for short) Smullyan invented to develop his recursive function theory, are proved suitable to generate languages. In this paper we first point out that EFS can also work as a logic programming language, and the resolution procedure for EFS can be used to accept languages. We give a theoretical foundation to EFS from the viewpoint of semantics of logic programs. Hence, Shapiro's theory of model inference can naturally be applied to our language learning by EFS. We introduce some subclasses of EFS's which correspond to Chomsky hierarchy and other important classes of languages. We discuss computations of unifiers between two terms. Then we give inductive inference algorithms including refinement operators for these subclasses and show their completeness.

1. Introduction

In computer science and artificial intelligence, learning or inductive inference is attracting much attention. Many contributions have been made in this field for the last 25 years [4]. Theoretical studies of language learning, originated in the so called grammatical inference, are now laying a firm foundation for the other approaches.

* This paper was supported by Grant-in-Aid for Scientific Research on Priority Areas (No. 63633011), The Ministry of Education, Science and Culture of Japan.

** Present address: Electrical Engineering Department, Hokkaido University, Sapporo 060, Japan.
to learning as the theory of languages and automata did for computer science in general [7, 1, 2, 4, 17]. However, most of such studies were developed in their own frameworks such as patterns, regular grammars, context-free and context-sensitive grammars, phrase structure grammars, many kinds of automata, and so on. Hence they had to devise also their own procedures for generating hypotheses from examples so far given and for testing each hypothesis on them.

In this paper we introduce variable-bounded EFS to language learning, especially to inductive inference of languages. The EFS, elementary formal system [20, 6], that was invented by Smullyan to develop his recursive function theory is also a good framework for generating languages [5].

Recently some new approaches to learning are proposed [16, 21, 3, 9] and being studied extensively [8, 14]. We here pay our attention to Shapiro's theory of model inference system (MIS for short) [16] that succeeded in unifying the various approaches to inductive inference such as program synthesis from examples, automatic knowledge acquisition, and automatic debugging. It has theoretical backgrounds in the first order logic and logic programming. His system also deals with language learning by using the so called difference-lists, which seem unnatural to develop the theory of language learning.

This paper combines EFS and MIS in order that we can take full advantage of theoretical results of them and extend our previous work [19]. First we give definitions of concepts necessary for our discussions. In Section 3 we show that the variable-bounded EFS has a good background in the theory of logic programming, and also it has an efficient derivation procedure for testing the guessed hypotheses on examples. In Section 4, we prove that the variable-bounded EFS's constitute a natural and proper subclass of the full EFS's, but they are powerful enough to define all the recursively enumerable sets of words. Then we describe in our framework many important subclasses of languages including Chomsky hierarchy and pattern languages. We also discuss the computations of unifiers which play a key role in the derivations for the above mentioned testing hypotheses. In Section 5 we give the inductive inference algorithms including contradiction backtracing and refinement operators for these subclasses in a uniform way, and prove their completeness. Thus our variable-bounded EFS works as an efficient unifying framework for language learning.

2. Preliminaries

Let $\Sigma, X,$ and $\Pi$ be mutually disjoint sets. We assume that $\Sigma$ and $\Pi$ are finite. We refer to $\Sigma$ as alphabet, and to each element of it as symbol, which will be denoted by $a, b, c, \ldots$, to each element of $X$ as variable, denoted by $x, y, z, x_1, x_2, \ldots$ and to each element of $\Pi$ as predicate symbol, denoted by $p, q, q_1, q_2, \ldots$, where each of them has an arity. $A^+$ denotes the set of all nonempty words over a set $A$. Let $S$ be an EFS that is being defined below.
Definition. A term of $S$ is an element of $(\Sigma \cup X)^*$. Each term is denoted by $\pi$, $\tau$, $\pi_1$, $\pi_2$, $\ldots$, $\tau_1$, $\tau_2$, $\ldots$. A ground term of $S$ is an element of $\Sigma^+$. Terms are also called patterns.

Definition. An atomic formula (or atom for short) of $S$ is an expression of the form $p(\tau_1, \ldots, \tau_n)$, where $p$ is a predicate symbol in $\Pi$ with arity $n$ and $\tau_1, \ldots, \tau_n$ are terms of $S$. The atom is ground if all $\tau_1, \ldots, \tau_n$ are ground.

Well-formed formulas, clauses, empty clause ($\Box$), ground clauses and substitutions are defined in the ordinary way [11].

Definition. A definite clause is a clause of the form $A \leftarrow B_1, \ldots, B_n$ ($n \geq 0$).

Definition (Smullyan [20]). An elementary formal system (EFS for short) $S$ is a triplet $(\Sigma, \Pi, \Gamma)$, where $\Gamma$ is a finite set of definite clauses. The definite clauses in $\Gamma$ are called axioms of $S$.

We denote a substitution by $\{x_i := \pi_1, \ldots, x_n := \pi_n\}$, where $x_i$ are mutually distinct variables. We also define $p(\tau_1, \ldots, \tau_n)\theta = p(\tau_1\theta, \ldots, \tau_n\theta)$ and

$$(A \leftarrow B_1, \ldots, B_m)\theta = A\theta \leftarrow B_1\theta, \ldots, B_m\theta,$$

for a substitution $\theta$, an atom $p(\tau_1, \ldots, \tau_n)$ and a clause $A \leftarrow B_1, \ldots, B_n$.

Definition. Let $S = (\Sigma, \Pi, \Gamma)$ be an EFS. We define the relation $\Gamma \vdash C$ for a clause $C$ of $S$ inductively as follows:

1. If $\Gamma \ni C$, then $\Gamma \vdash C$.
2. If $\Gamma \vdash C$ and $\Gamma \vdash C\theta$ for any substitution $\theta$.
3. If $\Gamma \vdash A \leftarrow B_1, \ldots, B_n$ and $\Gamma \vdash B_n \leftarrow$, then $\Gamma \vdash A \leftarrow B_1, \ldots, B_{n-1}$.

$C$ is provable from $\Gamma$ if $\Gamma \vdash C$.

Definition. For an EFS $S = (\Sigma, \Pi, \Gamma)$ and $p \in \Pi$ with arity $n$, we define $L(S, p) = \{(\alpha_1, \ldots, \alpha_n) \in (\Sigma^+)^n \mid \Gamma \vdash p(\alpha_1, \ldots, \alpha_n)\}$. In case $n = 1$, $L(S, p)$ is a language over $\Sigma$. A language $L \subseteq \Sigma^+$ is definable by EFS or an EFS language if such $S$ and $p$ exist.

Now we will give two interesting subclasses of EFS's. We need some notations. Let $v(\mathcal{E})$ be the set of all variables in $\mathcal{E}$, where $\mathcal{E}$ is an atom or a clause. For a term $\pi$, $|\pi|$ denotes the length of $\pi$, that is, the number of all occurrences of symbols and variables in $\pi$, and $o(x, \pi)$ denotes the number of all occurrences of a variable $x$ in term $\pi$. For an atom $p(\pi_1, \ldots, \pi_n)$, let

$$|p(\pi_1, \ldots, \pi_n)| = |\pi_1| + \cdots + |\pi_n|,$$

$$o(x, p(\pi_1, \ldots, \pi_n)) = o(x, \pi_1) + \cdots + o(x, \pi_n).$$
**Definition.** A definite clause \( A \leftarrow B_1, \ldots, B_n \) is *variable-bounded* if \( v(A) \supseteq v(B_i) \) \((i = 1, \ldots, n)\), and an EFS is *variable-bounded* if its axioms are all variable-bounded.

**Definition.** A clause \( A \leftarrow B_1, \ldots, B_n \) is *length-bounded* if 
\[
|A\theta| \geq |B_1\theta| + \cdots + |B_n\theta|
\]
for any substitution \( \theta \). An EFS \( S = (\Sigma, \Pi, \Gamma) \) is *length-bounded* if axioms in \( \Gamma \) are all length-bounded.

We can easily characterize the concept of length-boundedness as follows.

**Lemma 2.1.** A clause \( A \leftarrow B_1, \ldots, B_n \) is length-bounded if and only if 
\[
|A| \geq |B_1| + \cdots + |B_n|,
\]
\[
o(x, A) \geq o(x, B_1) + \cdots + o(x, B_n)
\]
for any variable \( x \).

**Proof.** Let \( A \leftarrow B_1, \ldots, B_n \) be a length-bounded clause. Then 
\[
|A\theta| \geq |B_1\theta| + \cdots + |B_n\theta|
\]
for any substitution \( \theta \). When \( \theta = \{ \} \), we have 
\[
|A| \geq |B_1| + \cdots + |B_n|.
\]
Let \( \theta = \{ x := x^{k+1} \} \). Then 
\[
|A\theta| - \sum_{i=1}^{n} |B_i\theta| = |A| - \sum_{i=1}^{n} |B_i| + k \times \left( o(x, A) - \sum_{i=1}^{n} o(x, B_i) \right) \geq 0.
\]
Therefore 
\[
o(x, A) - \sum_{i=1}^{n} o(x, B_i) \geq \frac{-(|A| - \sum_{i=1}^{n} |B_i|)}{k}.
\]
If \( k \) is large enough, for example, \( k > |A| - \sum_{i=1}^{n} |B_i| \), we have 
\[
o(x, A) - \sum_{i=1}^{n} o(x, B_i) \geq 0.
\]
Conversely let \( A, B_1, \ldots, B_n \) be atoms such that 
\[
|A| \geq |B_1| + \cdots + |B_n|,
\]
\[
o(x, A) \geq o(x, B_1) + \cdots + o(x, B_n)
\]
for any variable \( x \), and let \( \theta \) be any substitution. Then 
\[
|A\theta| - \sum_{i=1}^{n} |B_i\theta| = |A| + \sum_{x \in v(A)} ((|x\theta| - 1) o(x, A))
\]
\[
- \sum_{i=1}^{n} \left( |B_i| + \sum_{x \in v(A)} ((|x\theta| - 1) o(x, B_i)) \right)
\]
\[
= |A| - \sum_{i=1}^{n} |B_i| + \sum_{x \in v(A)} \left( (|x\theta| - 1) \left( o(x, A) - \sum_{i=q}^{n} o(x, B_i) \right) \right) \geq 0.
\]
Here we should note that $|x\theta| \geq 1$ for any substitution. In case we allow an erasing substitution $\theta$ such that $|x\theta| = 0$, this lemma does not hold. 

By this lemma we know that length-bounded clauses are all variable-bounded and it is computable to test whether a given clause is length-bounded or not.

**Example 2.1.** An EFS $S = (\{a, b, c\}, \{p, q\}, \Gamma')$ with
\[
\Gamma' = \begin{cases}
p(a, b, c) \leftarrow, \\
p(ax, by, cz) \leftarrow p(x, y, z), \\
q(xyz) \leftarrow p(x, y, z)
\end{cases}
\]
is variable-bounded, and also length-bounded by Lemma 2.1. It defines a language $L(S, q) = \{a^n b^n c^n | n \geq 1\}$.

### 3. EFS as a logic programming language

In this section we show that EFS is a logic programming language. We give a refutation procedure for EFS and several kinds of semantics for EFS. Then we show that the refutation is complete as a procedure to accept EFS languages. We also show that the negation as a failure rule for variable-bounded EFS is complete and it is coincident with the Herbrand rule.

#### 3.1. Derivation procedure for EFS

**Definition.** Let $\alpha$ and $\beta$ be a pair of terms or atoms. Then a substitution $\theta$ is a unifier of $\alpha$ and $\beta$ if $\alpha \theta = \beta \theta$.

It is often the case that there are infinitely many maximally general unifiers.

**Example 3.1** (Plotkin [13]). Let $S = (\{a, b\}, \{p\}, \Gamma)$.

- For every $i$, $\{x := a^i\}$ is the unifier of $p(ax)$ and $p(xa)$. All the unifiers are maximally general.

We formalize the derivation for an EFS with no requirement that every unifier should be most general.

**Definition.** A goal clause (or goal for short) of $S$ is a clause of the form
\[
\leftarrow B_1, \ldots, B_n \quad (n \geq 0).
\]

**Definition.** If clauses $C$ and $D$ are identical except renaming of variables, that is, $C = D\theta$ and $C\theta' = D$ for some substitutions $\theta$ and $\theta'$, we say $D$ is a variant of $C$ and write $C \equiv D$. 

We assume a computation rule $R$ to select an atom from every goal.

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

1. $G_i$ is a goal, $\theta_i$ is a substitution, $C_i$ is a variant of an axiom of $S$, and $G_0 = G$.
2. $v(C_i) \cap v(C_j) = \emptyset$ for every $i$ and $j$ such that $i \neq j$, and $v(C_i) \cap v(G_i) = \emptyset$ for every $i$.
3. If $G_i$ is $\leftarrow A_1, \ldots, A_k$ and $A_m$ is the atom selected by $R$, then $C_i$ is $A \leftarrow B_1, \ldots, B_q$, and $\theta_i$ is a unifier of $A$ and $A_m$, and $G_{i+1}$ is

   $$\leftarrow A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k, \theta_i.$$ 

   $A_m$ is a selected atom of $G_i$, and $G_{i+1}$ is a resolvent of $G_i$ and $C_i$ by $\theta_i$.

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

**Example 3.2.** Let EFS $S = (\{a, b\}, \{p\}, \Gamma)$ with

$$\Gamma = \begin{cases} p(a) \leftarrow, \\
p(bxy) \leftarrow p(x), p(y) \end{cases}.$$ 

Then a refutation from $\leftarrow p(baaba)$ is illustrated by Fig. 1, where the computation rule selects the leftmost atom from every goal.

Now we give a property of unification. Makanin [12] showed that the existence of a unifier of two terms is decidable, but this fact is not sufficient for constructing derivations. For ground patterns we have a good property.

![Fig. 1. A refutation.](image-url)
Lemma 3.1 (Yamamoto [23]). Let $\alpha$ and $\beta$ be a pair of terms or atoms. If one of them is ground, then every unifier of $\alpha$ and $\beta$ is ground and the set of all unifiers is finite and computable.

The aim of our formalization of derivation is to give a procedure accepting languages definable by EFS's. We will show in Section 4 that the variable-bounded EFS's are powerful enough. Thus we can assume that every derivation starts from a ground goal and that every EFS is variable-bounded. Then we get the following lemma directly from Lemma 3.1 and the definition of variable-bounded clauses.

Lemma 3.2 (Yamamoto [23]). Let $S$ be a variable-bounded EFS, and $G$ be a ground goal. Then every resolvent of $G$ is ground, and the set of all the resolvents of $G$ is finite and computable.

This lemma shows that we can implement the derivation for variable-bounded EFS in nearly the same way as in the traditional logic programming languages.

If we do not have the assumption above, we need an alternative formalization of derivation, such as given by Yamamoto [22], to control the unification which is not always terminating.

3.2. Completeness of refutation

We describe the semantics of EFS's according to Jaffar et al. [10]. They have given a general framework of various logic programming languages by representing their unification algorithm as an equality theory. To represent the unification in the refutation for EFS we use the equality theory

$$E = \{\text{cons}(\text{cons}(x, y), z) = \text{cons}(x, \text{cons}(y, z))\},$$

where $\text{cons}$ is to be interpreted as the catenation of terms.

The first semantics for an EFS $S = (\Sigma, \Pi, \Gamma)$ is its model. To interpret well-formed formulas of $S$ we can restrict the domains to the models of $E$. Then a model of $S$ is an interpretation which makes every axiom in $\Gamma$ true. We can use the set of all ground atoms as the Herbrand base denoted by $B(S)$. Every subset $I$ of $B(S)$ is called an Herbrand interpretation in the sense that $A \in I$ means $A$ is true and $A \notin I$ means $A$ is false for $A \in B(S)$. Then

$$M(S) = \bigcap \{M \subseteq B(S) \mid M \text{ is an Herbrand model of } S\}$$

is an Herbrand model of $S$, and every ground atom in $M(S)$ is true in any model of $S$. The second semantics is the least fixpoint $\text{lfp}(T_S)$ of the function $T_S : 2^{B(S)} \rightarrow 2^{B(S)}$ defined by

$$T_S(I) = \{A \in B(S) \mid \text{there is a ground instance } A \leftarrow B_1, \ldots, B_n \text{ of an axiom of } S \text{ such that } B_k \in I \text{ for all } k \ (1 \leq k \leq n)\}.$$
\[ \begin{array}{c}
\text{\textit{p(baa)}} \quad \text{\textit{p(bxy)}} \quad \text{\textit{p(x), p(y)}} \\
\text{\textit{x := a, y := aa}} \\
\text{\textit{p(a), p(aa)}} \\
\text{\textit{p(a)\leftarrow}} \\
\text{\textit{p(aa)\leftarrow failed!}}
\end{array} \]

Fig. 2. A derivation finitely failed with length 2.

\( \text{lfp}(T_S) \) is identical to \( T_S \uparrow \omega \) defined as follows:

\[
\begin{align*}
T_S \uparrow 0 & = \emptyset, \\
T_S \uparrow n & = T_S(T_S \uparrow (n-1)) \quad \text{for } n \geq 1, \\
T_S \uparrow \omega & = \bigcup_{n=0} \n T_S \uparrow n.
\end{align*}
\]

The third semantics using refutation is defined by

\[
\text{SS}(S) = \{ A \in B(S) | \text{there exists a refutation from } \leftarrow A \}.
\]

These three semantics are shown to be identical by Jaffar et al. [10].

Now we give another semantics of EFS using the provability as the set

\[
\text{PS}(S) = \{ A \in B(S) | \Gamma \vdash A \}.
\]

**Theorem 3.1** (Yamamoto [23]). *For every EFS S, \( M(S) = \text{lfp}(I_S) = I_S \uparrow \omega = \text{SS}(S) = \text{PS}(S) \).*

Thus the refutation is complete as a procedure to accept EFS languages.

### 3.3. Negation as failure for EFS

Now we discuss the inference of negation. We start with some definitions.

**Definition.** A derivation is \emph{finitely failed with length n} if its length is \( n \) and there is no axiom which satisfies condition (3.3) for the selected atom of the last goal.

**Example 3.3.** Let \( S \) be the EFS in Example 3.2. Then the derivation illustrated in Fig. 2 is finitely failed with length 2.

**Definition.** A derivation \( (G_i, \theta_i, C_i) \) \( (i = 0, 1, \ldots) \) is \emph{fair} if it is finitely failed or, for each atom \( A \) in \( G_i \), there is a \( k \geq i \) such that \( A \theta_i \ldots \theta_{k-1} \) is the selected atom of \( G_k \).

In the discussion of negation, we assume that any computation rule \( R \) makes all derivations \emph{fair}. We say such a computation rule to be \emph{fair}.
The negation as failure rule is the rule that infers \( \neg A \) when a ground atom \( A \) is in the set

\[
FF(S) = \{ A \in B(S) \mid \text{for any fair computation rule, there is an } n \text{ such that all derivations from } \leftarrow A \text{ are finitely failed within length } n \}.
\]

Put \( ecj(\theta) = (x_1 = \tau_1 \land \cdots \land x_n = \tau_n) \) for a substitution \( \theta = \{ x_1 := \tau_1, \ldots, x_n := \tau_n \} \), and for an empty \( \theta, ecj(\theta) = true. \) By Jaffar et al. [10], negation as failure for EFS is complete if the following two are satisfied:

1. \( \exists \text{ a theory } E^* \text{ such that, for every two terms } \pi \text{ and } \tau, (\pi = \tau) \rightarrow \bigvee_{i=1}^{k} ecj(\theta_i) \) is a logical consequence, where \( \theta_1, \ldots, \theta_k \) are all unifiers of \( \pi \) and \( \tau \), and the disjunction means \( \Box \) if \( k = 0 \).
2. \( GF(S) = \{ A \in B(S) \mid \text{for any fair computation rule, all derivations from } \leftarrow A \text{ are finitely failed} \} \)

In general, we can easily construct an EFS such that \( FF(S) \neq GF(S) \).

We show that the negation as failure rule for variable-bounded EFS is complete. To prove the completeness, we need the set

\[
GGF(S) = \{ A \in B(S) \mid \text{for any fair computation rule, all derivations from } \leftarrow A \text{ such that all goals in them are ground are finitely failed} \}.
\]

The inference rule that infers \( \neg A \) for a ground atom \( A \) if \( A \) is in \( GGF(S) \) is called the Herbrand rule [11].

**Theorem 3.2** (Yamamoto [23]). For any variable-bounded EFS \( S \),

\[
FF(S) = GF(S) = GGF(S).
\]

By this theorem we can use the following equality theory instead of (3.4):

\[
E^* = \{ \pi = \tau \rightarrow \bigvee_{i=1}^{k} ecj(\theta_i) \mid \pi \text{ is a ground term, } \tau \text{ is a term, and } \theta_1, \ldots, \theta_k \text{ are all unifiers of } \pi \text{ and } \tau \}.
\]

Thus the negation as failure rule is complete and identical to the Herbrand rule for variable-bounded EFS's. Yamamoto [23] has discussed the closed world assumption for EFS.

4. The classes of EFS languages

We describe the classes of our languages comparing with Chomsky hierarchy and some other classes. Throughout the paper we do not deal with the empty word.
4.1. The power of EFS

The first theorem shows the variable-bounded EFS's are powerful enough.

**Theorem 4.1.** Let \( \Sigma \) be an alphabet with at least two symbols. Then a language \( L \subseteq \Sigma^+ \) is definable by a variable-bounded EFS if and only if \( L \) is recursively enumerable.

**Proof.** A Turing machine with left and right endmarkers to indicate the both ends of currently used tape can be simulated in a variable-bounded EFS by encoding tape symbols to words of \( \Sigma^+ \). The converse is clear from Smullyan [20]. □

The left to right part of Theorem 1.4 is still valid in case alphabet \( \Sigma \) is a singleton. However, to show the converse we need to weaken the statement slightly just as in Theorem 4.2(2) below, or to simulate two-way counter machines.

Now we show relations between length-bounded EFS and CSG.

**Theorem 4.2.** (1) Any length-bounded EFS language is context-sensitive.

(2) For any context-sensitive language \( L \subseteq \Sigma^+ \), there exist a superset \( \Sigma_0 \) of \( \Sigma \), a length-bounded EFS \( S = (\Sigma_0, \Pi, \Gamma) \) and \( p \in \Pi \) such that \( L = L(S, p) \cap \Sigma^+ \).

**Proof.** (1) Any derivation in a length-bounded EFS from a ground goal can be simulated by a nondeterministic linear bounded automaton, because all the goals in the derivation are kept ground and the total length of the newly added subgoals in each resolution step does not exceed the length of the selected atom by the definition.

(2) This can also be proved by a simulation. □

The set \( \Sigma_0 - \Sigma \) above corresponds to the auxiliary alphabet like tape symbols or nonterminal symbols. We can show another theorem related to the converse of Theorem 4.2(1).

**Definition.** A function \( \sigma \) from \( \Sigma^+ \) into itself is **length-bounded EFS realizable** if there exist a length-bounded EFS \( S_0 = (\Sigma, \Pi_0, \Gamma_0) \) and a binary predicate symbol \( p \in \Pi_0 \) for which \( \Gamma_0 \vdash p(u, w) \iff w = \sigma(u) \).

**Theorem 4.3.** Let \( \Sigma \) be an alphabet with at least two symbols. Then for any context-sensitive language \( L \subseteq \Sigma^+ \), there exist a length-bounded EFS \( S = (\Sigma, \Pi, \Gamma') \), a length-bounded EFS realizable function \( \sigma \) and \( p \in \Pi \) associated with \( \sigma \) such that

\[
L = \{w \in \Sigma^+ \mid \Gamma \vdash p(w, \sigma(w))\}.
\]

**Proof.** Let \( \Sigma = \{a_1, \ldots, a_s\} \), and \( T = \{a_1, \ldots, a_n\} \) be the tape symbols of the linear bounded automaton \( M \) which accepts \( L \), where \( 1 < s \leq n \). Let \( a_1 = 0 \) and \( a_2 = 1 \). We define the function \( \sigma \) as a homomorphism on \( (T \cup \{\dagger\})^* \) by

\[
\sigma(a_i) = 1i_1 \ldots i_k \quad (1 \leq i \leq n), \quad \sigma(\dagger) = 10^k,
\]
where \( k \) is an integer such that \( 2^{k-1} \leq n < 2^k \), \( \uparrow \) denotes the head position in the tape of \( M \), and \( i_1 \ldots i_k \) is the \( k \)-figure binary notation of \( i \). Then the \( \sigma \) is clearly length-bounded EFS realizable. We can easily simulate the \( M \) on the second arguments of \( p(w, \sigma(w)) \).

### 4.2. Smaller classes of EFS languages

Now we compare EFS languages with some other smaller classes of languages.

**Definition.** A length-bounded EFS \( S = (\Sigma, \Pi, \Gamma) \) is simple if \( \Pi \) consists of unary predicate symbols and each axiom in \( \Gamma \) is of the form

\[
p(\pi) \leftarrow q_1(x_1), \ldots, q_n(x_n),
\]

where \( x_1, \ldots, x_n \) are mutually distinct variables.

**Example 4.1.** An EFS \( S = (\{a\}, \{p\}, \Gamma) \) with

\[
\Gamma = \begin{cases} 
p(a) \leftarrow, 
\quad p(x) \leftarrow p(x), 
\end{cases}
\]

is simple and \( L(S, p) = \{a^n \mid n \geq 0\} \).

It is known that simple EFS languages are context-sensitive \([5]\).

**Definition.** A pattern \( \pi \) is regular if \( o(x, \pi) = 1 \) for any variable \( x \). A simple EFS \( S = (\Sigma, \Pi, \Gamma) \) is regular if the pattern in the head of each definite clause in \( \Gamma \) is regular.

**Example 4.2.** An EFS \( S = (\{a, b\}, \{p\}, \Gamma) \) with

\[
\Gamma = \begin{cases} 
p(ab) \leftarrow, 
\quad p(axb) \leftarrow p(x), 
\end{cases}
\]

is regular and \( L(S, p) = \{a^n b^n \mid n \geq 1\} \).

**Theorem 4.4.** A language is definable by a regular EFS if and only if it is context-free.

**Definition.** A regular EFS \( S = (\Sigma, \Pi, \Gamma) \) is right-linear (left-linear) if each axiom in \( \Gamma \) is of one of the following forms:

\[
p(\pi) \leftarrow, \quad p(ux) \leftarrow q(x) \quad (p(xu) \leftarrow q(x)),
\]

where \( \pi \) is a regular pattern and \( u \in \Sigma^+ \).

A regular EFS is one-sided linear if it is right- or left-linear.
Theorem 4.5. A language is definable by a one-sided linear EFS if and only if it is regular.

These two theorems can easily be proved by noticing that a production rule, say
\[ p \to uqr \]
of a context free grammar can be transformed into a clause
\[ p(uxy) \leftarrow q(x), r(y) \]
of the regular EFS, where \( p, q \) and \( r \) are nonterminals and \( u \) is a terminal string, and we confuse the nonterminals and predicate symbols.

The pattern languages [1, 2, 17, 18] which are important in inductive inference of languages from positive data are also definable by special simple EFS's.

4.3. Computations of unifiers

As we have stated in Section 3, all the goals in the derivation from a ground goal are kept ground, because we deal with only the variable-bounded EFS's. Hence, every unification is made between a term and a ground term. To find a unifier is to get a solution of equation \( w = \pi \), where \( w \) is a ground term and \( \pi \) is a term possibly with variables. In general, as is easily seen, the equation can be solved in \( O(|w|^{|\pi|}) \) time. Hence, for a fixed EFS, it can be solved in time polynomial in the length of the ground goal. However, if the EFS is not fixed, the problem is NP-complete, because it is equivalent to the membership problem of pattern languages [1].

As for the one-sided linear and regular EFS's, the problem can be proved to have good properties.

Proposition 4.1. The equation \( w = \pi \) has at most one solution for every \( w \in \Sigma^+ \) if and only if \( \pi \) contains at most one variable.

Proposition 4.2 (Shinohara [17]). Let \( w \) be a word in \( \Sigma^+ \) and \( \pi \) be a regular pattern. Then each unifier of \( w \) and \( \pi \) is computed in \( O(|w| + |\pi|) \) time.

By these propositions, the unifier of \( w \) and \( \pi \) is at most unique in one-sided linear EFS, and each unifier of them can be computed in a linear time in regular EFS. However, in the worst case, there may exist unifiers in regular EFS as many as \( |w|^{|\pi|} \).

5. Inductive inference of EFS languages

In this section, we show how EFS languages are inductively learned. To specify inductive inference problems we need to give five items, the set of rules, the representation of rules, the data presentation, the method of inference called the inference machine, and the criterion of successful inference [4].
In our problem, the class of rules are EFS languages. The examples are ground atoms $A$ with sign $+$ or $-$ indicating whether $A$ is provable from the target EFS or not. An example $+A$ is said to be positive, $-A$ negative. Our criterion of successful inference is the traditional identification in the limit [7].

The inference machine we consider here is based on Shapiro's MIS (Model Inference System) [16]. The following procedure MIEFS (Model Inference for EFS) describes the outline of our inference method, which uses a subprocedure CBA (Contradiction Backtracing Algorithm) and refinements of clauses. The hypothesis $H$ is too strong, if $H$ proves $A$ for some negative example $-A$. $H$ is too weak, if $H$ can not prove $A$ for some positive example $+A$.

When MIEFS finds the current hypothesis $H$ is not compatible with the examples read so far, it tries to modify $H$ as follows. If $H$ is too strong, then MIEFS searches $H$ for a false clause $C$ by using CBA and deletes $C$ from $H$. Otherwise MIEFS increases the power of $H$ by adding refinements of clauses deleted so far. A refinement $C'$ of a clause $C$ is a logical consequence of $C$. Therefore the hypothesis obtained by adding a refinement $C'$ is weaker than the hypothesis before deleting $C$.

**Procedure MIEFS;**

begin

$H := \{\square\}$;

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 $H$;

delete $C$ from $H$;

end

while $H$ is too weak do

add a refinement of clause deleted so far to $H$;

end

output $H$;

forever

end

To guarantee our procedure MIEFS successfully identifies EFS languages, it is necessary to test whether CBA works for EFS's or not, and to devise refinement operator and show its completeness.

5.1. Contradiction backtracing algorithm for EFS

Contradiction backtracing algorithm (CBA for short) devised by Shapiro [16] makes use of a refutation indicating a hypothesis $H$ is too strong. It traces selected atoms backward in the refutation. By using an oracle ASK, it tests their truth values to detect a false clause in $H$. When $A$, is not ground, CBA must select a ground
instance of $A_i$. However, in variable-bounded EFS's, $A_i$ is always ground, and hence we can simplify CBA as follows.

Procedure CBA for EFS;

Input: $(G_0 = G, \theta_0, C_0), (G_1, \theta_1, C_1), \ldots, (G_k = \Box, \theta_k, C_k)$; {a refutation of a ground goal $G$ true 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 $\text{ASK}(A_i)$ is false then return $C_{i-1}$;
end
end

The following lemma and theorem show our CBA procedure works correctly.

Lemma 5.1. Let $G'$ be the resolvent of a ground goal $G$ and a variable-bounded clause $C$ by a substitution $\theta$ and $A$ be the selected atom of $G$. Assume that $G'$ is false in a model $M$. If $A$ is true in $M$ then $G$ is false in $M$. Otherwise $C \theta$ is ground and false in $M$.

Proof. Let $G = \leftarrow A_1, \ldots, A_n$ be a ground goal and $C = A' \leftarrow B_1, \ldots, B_q$ be a variable-bounded clause, where $A = A_m$. Then

$$G' = \leftarrow A_1, \ldots, A_m, B_1 \theta, \ldots, B_q \theta, A_{m+1}, \ldots, A_n$$

is a ground resolvent of $G$ and $C$. Since we assume $G'$ is false in a model $M$, all atoms in $A_1, \ldots, A_m, A_{m+1}, \ldots, A_n$ and $B_1 \theta, \ldots, B_q \theta$ are ground and true in $M$. Therefore if $A$ is true in $M$, then $G = \leftarrow A_1, \ldots, A_m, A, A_{m+1}, \ldots, A_n$ is false in $M$, otherwise $C \theta = A \leftarrow B_1 \theta, \ldots, B_q \theta$ is false in $M$. $\square$

Theorem 5.1. Let $M$ be a model of a variable-bounded EFS $S$, and $(G_0 = G, \theta_0, C_0), (G_1, \theta_1, C_1), \ldots, (G_k = \Box, \theta_k, C_k)$ be a refutation by $S$ of a ground goal $G$ true in $M$. If CBA is given the refutation, then it makes $i$ oracle calls and returns $C_{k-i}$ false in $M$ for some $i = 1, 2, \ldots, k$.

Proof. By Lemma 5.1 and an induction on $k - i$, the number of oracle calls made by CBA, we can easily prove that the clause returned by CBA is false in $M$.

We may assume that $G_0$ is not empty. Hence $k - i$ is positive. If CBA makes the $k$th call to the oracle ASK, then the received truth value of $A_i$ upon which $G_i$ is resolved must be false because $A_i$ is identical to an atom in $G_0$. Therefore CBA always returns a clause $C_{k-i}$ after making at most $k$ oracle calls. $\square$
5.2. Refinement operator for EFS

We assume a structural complexity measure size of patterns and clauses such that the number of patterns or clauses whose sizes are equal to \( n \) is finite (except renaming of variables) for any integer \( n \). In what follows, we identify variants with each other.

**Definition.** We define the size of an atom \( A \) by

\[
\text{size}(A) = 2 \times |A| - \#v(A)
\]

where \( \#S \) is the number of elements in a set \( S \). For a clause \( C = A \leftarrow B_1, \ldots, B_n \), we define

\[
\text{size}(C) = 2 \times (|A| + |B_1| + \cdots + |B_n|) - \#v(C).
\]

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

Now we introduce refinement operators for the subclasses of EFS's. All refinement operators defined below have a common feature. They are constructed by two types of operations, applying a substitution and adding a literal.

**Definition.** A substitution \( \theta \) 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 \( y \neq z \).

**Lemma 5.2.** Let \( \theta \) be a basic substitution for a clause \( C \). Then \( \text{size}(C) < \text{size}(C\theta) \).

**Proof.** If \( \theta \) is of the form \( \{x := y\} \) or \( \{x := a\} \), then \( \#v(C\theta) = \#v(C) - 1 \). Therefore \( \text{size}(C\theta) = \text{size}(C) + 1 \). If \( \theta \) is of the form \( \{x := yz\} \), then \( |C\theta| = |C| + o(x, C) \) and \( \#v(C\theta) - \#v(C) = 1 \). Since \( o(x, C) \geq 1 \), \( \text{size}(C\theta) = \text{size}(C) + 2 \times o(x, C) - 1 \). □

**Definition.** Let \( A \) be an atom. Then an atom \( B \) in \( \rho_\downarrow(A) \) if and only if

1. \( A = \square \) and \( B = p(x_1, \ldots, x_n) \) for a predicate symbol \( p \) with arity \( n \) and mutually distinct variables \( x_1, \ldots, x_n \), or
2. \( A\theta = B \) for a substitution \( \theta \) basic for \( A \).

**Lemma 5.3.** Let \( C \) and \( D \) be clauses such that \( C\theta = D \) but \( C \neq D \) for some substitution \( \theta \). Then there exists a sequence of substitutions \( \theta_1, \theta_2, \ldots, \theta_n \) such that \( \theta_i \) is basic for \( C\theta_1 \ldots \theta_{i-1} (i = 1, \ldots, n) \) and \( C\theta_1 \ldots \theta_n = D \).
Theorem 5.2. \( \rho_a \) is a locally finite and complete refinement operator for atoms.

Shinohara [17] discussed inductive inference of pattern languages from positive data. The method he called tree search method uses a special version of the refinement operator \( \rho_a \). His method first tries to apply substitutions of type \( \{ x := yz \} \) to get the longest possible pattern, and then tries to apply substitutions of type \( \{ x := a \} \), and finally tries to unify variables by substitutions of type \( \{ x := y \} \).

**Definition.** Let \( C \) be a variable-bounded clause. Then a clause \( D \) is in \( p_{Vb}(C) \) if and only if (5.4) or (5.5) holds, or \( C = A \leftarrow B_1, \ldots, B_{n-1} \) and \( D = A \leftarrow B_1, \ldots, B_{n-1}, B_n \) is variable-bounded.

Similarly we define \( p_{lb} \) for length-bounded clauses.

Theorem 5.3. \( \rho_{vb} \) is a complete refinement operator for variable-bounded clauses.

Theorem 5.4. \( \rho_{lb} \) is a locally finite and complete refinement operator for length-bounded clauses.

Note that \( p_{vb} \) is not locally finite because the number of atoms \( B_n \) possibly added by \( \rho_{vb} \) is infinite, while \( \rho_{lb} \) is locally finite. We can also define refinement operators for simple or regular clauses and prove they are locally finite and complete. For simple clauses, applications of basic substitutions should be restricted only to atoms. Further, for regular clauses, substitutions of the form \( \{ x := y \} \) should be inhibited.

6. Conclusion

We have introduced several important subclasses of EFS's by gradually imposing restrictions on the axioms, and given a theoretical foundation of EFS's from the viewpoint of logic programming. EFS's work for accepting languages as well as for generating them. This aspect of EFS's is particularly useful for inductive inference of languages. We have also shown inductive inference algorithms for some subclasses of EFS's in a uniform way and proved their completeness. Thus, EFS's are a good unifying framework for inductive inference of languages.

We can introduce pairs of parentheses to simple EFS's just like parenthesis grammars. Nearly the same approaches as [24, 15] will be applicable to our inductive inference of simple EFS languages. Thus, we can resolve the computational hardness of unifications.

There are many other problems in connection with computational complexity, the learning models such as [3, 21], and introduction of the empty word [18] which we will discuss elsewhere.
References