Elementary Formal System as a Unifying Framework for Language Learning

Arikawa, Setsuo
Research Institute of Fundamental Information Science Kyushu University

Shinohara, Takeshi
Department of Artificial Intelligence Kyushu Institute of Technology

Yamamoto, Akihiro
Department of Information Systems, Kyushu University

https://hdl.handle.net/2324/3120

出版情報：RIFIS Technical Report. 14, 1989-03-15. Research Institute of Fundamental Information Science, Kyushu University
バージョン：
権利関係：

RIFIS-TR-CS-14

RIFIS Technical Report

Elementary Formal System as a Unifying Framework for Language Learning

Setsuo Arikawa, Takeshi Shinohara and Akihiro Yamamoto

March 15, 1989

Research Institute of Fundamental Information Science
Kyushu University 33
Fukuoka 812, Japan
E-mail: arikawa@rifis1.sci.kyushu-u.ac.jp Phone: 092(641)1101 Ext.4470

Elementary Formal System as a Unifying Framework for Language Learning

(Extended Abstract)*

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

Takeshi Shinohara
Department of Artificial Intelligence
Kyushu Institute of Technology, Iizuka 820

Akihiro Yamamoto
Department of Information Systems
Kyushu University 39, Kasuga 816

Abstract

This paper presents a unifying framework for language learning, especially for inductive inference of various classes of languages. 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 the computations of unifiers between two terms. Then we give, in a uniform way, 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 to learning as the theory of languages and automata did for Computer Science in general [7, 1, 2, 4, 14]. However,

*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.

most of such studies were developed in their own frameworks such as patterns, regular grammars, context-free and -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 a new unifying framework called variable-bounded EFS to language learning, especially to inductive inference of languages. The EFS, elementary formal system [17, 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 [13, 18, 3, 8] and being studied extensively as seen, for example, in COLT'88. We here pay our attention to Shapiro's theory of model inference system (MIS for short) [13] 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 seems 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 [16]. 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 are 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 we conclude that our variable-bounded EFS is an efficient unifying framework for language learning.

2 Preliminaries

Let Σ, X, and Π be mutually disjoint sets. We assume that Σ and Π are finite . We refer to Σ 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 Π 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 Σ^+. 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 Π with arity n and $\tau_1, \ldots, \tau_n$ are terms of S. The atom is *ground* if all the $\tau_1, \ldots, \tau_n$ are ground.

Well-formed formulas, clauses, empty clause ($\Box$), *ground clauses* and *substituions* are defined in the ordinal ways [10].

Definition (Smullyan [17]) An *elementary formal system* (*EFS* for short) S is a triplet (Σ, Π, Γ), where Γ is a finite set of definite clauses. The definite clauses in Γ are called *axioms* of S.

We denote a substitution by $\{x_1 := \tau_1, \ldots, x_n := \tau_n\}$, 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 θ, an atom $p(\tau_1, \ldots, \tau_n)$ and a clause $A \leftarrow B_1, \ldots, B_m$.

Definition A clause C of an EFS $S = (\Sigma, \Pi, \Gamma)$ is *provable*, denoted by $\Gamma \vdash C$, if it is either an axiom in Γ, or obtained from the axioms in Γ by finite applications of substitutions and modus ponens.

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 Σ. 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 $v(A)$ be the set of all variables in an atom A. For a term π, $|\pi|$ denotes the length of π, that is, the number of all occurrences of symbols and variables in π, and $o(x,\pi)$ denotes the number of all occurrences of a variable x in a term π. For an atom $p(\pi_1, \ldots, \pi_n)$, let

$$\begin{aligned} |p(\pi_1, \ldots, \pi_n)| &= |\pi_1| + \cdots + |\pi_n|, \\ o(x, p(\pi_1, \ldots, \pi_n)) &= o(x, \pi_1) + \cdots + o(x, \pi_n). \end{aligned}$$

Definition A definite clause $A \leftarrow B_1, \ldots, B_n$ is *variable-bounded* if $v(A) \supset v(B_i)$ $(i = 1, ..., n)$, and an EFS is *variable-bounded* if its axioms are all variable-bounded.

Definition A variable-bounded EFS $S = (\Sigma, \Pi, \Gamma)$ is *length-bounded* if

$$|A| \geq |B_1| + \cdots + |B_k|, \;\; o(x, A) \geq o(x, B_1) + \cdots + o(x, B_k)$$

for any $x \in v(A)$ and any axiom $A \leftarrow B_1, \ldots, B_k$in Γ.

Example 2.1 An EFS $S = (\{a,b,c\}, \{p,q\}, \Gamma)$ with

$$\Gamma = \{p(a,b,c) \leftarrow, \quad p(ax, by, cz) \leftarrow p(x,y,z), \quad q(xyz) \leftarrow p(x,y,z)\}$$

is length-bounded and defines a language $L(S,q) = \{a^n b^n c^n \mid n \geq 1\}$.

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

Lemma 2.1 *A variable-bounded EFS $S = (\Sigma, \Pi, \Gamma)$ is length-bounded if and only if*

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

for any axiom $A \leftarrow B_1, \ldots, B_n$ in Γ and any substitution θ.

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 show its completeness by giving several kinds of semantics for EFS. Then we show that negation as failure rule for variable-bounded EFS is complete and it is coincident with Herbrand rule.

3.1 Derivation procedure for EFS

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

If terms π and τ are identical except renaming of variables, that is, $\pi = \tau\theta$ and $\pi\theta' = \tau$ for some substitutions θ and θ', we say π is a *variant* of τ and write $\pi \equiv \tau$. Similarly we write $C \equiv D$ for clauses.

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

Example 3.1 (Plotkin[11]) Let $S = (\{a, b\}, \{p\}, \Gamma)$. Then $\{x := a^i\}$ for every i is the unifier of $p(ax)$ and $p(xa)$. All the unifiers are maximally general. In case $\Sigma = \{a\}$, the empty substitution is the most general unifier of the two atoms.

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

$$\leftarrow B_1, \ldots, B_n \qquad (n \geq 0).$$

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, θ_i, C_i) $(i = 0, 1, \ldots)$ which satisfies the following conditions:

(3.1) G_i is a goal, θ_i is a substitution, C_i is a variant of an axiom of S, and $G_0 = G$.

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

(3.3) 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 θ_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 θ_i.

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

To control the nondeterministic algorithm of unification, Yamamoto [19] gave another formulation of the derivation. We can overcome the problem with the following proposition.

Proposition 3.1 *Let α and β be a pair of terms or atoms. If one of them is ground, then every unifier of α and β is ground and the set of all unifiers is finite and computable.*

By this proposition, we can make the unification deterministic for variable-bounded EFS's because every goal in a derivation from a ground goal is ground. It suffices to deal with variable-bounded EFS's and ground goals for our discussions. We can implement the derivation in nearly the same way as in the traditional logic programming languages.

3.2 Completeness of refutation

We describe the semantics of refutation according to Jaffar, et al.[9]. 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

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

where *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 Γ 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 *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

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

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 $lfp(T_S)$ of the function $T_S : 2^{B(S)} \longrightarrow 2^{B(S)}$ defined by

$$T_S(I) = \left\{ A \in B(S) \;\middle|\; \begin{array}{l} \text{there is a ground instance of an axiom} \\ \qquad A \leftarrow B_1, \ldots, B_n \\ \text{of } S \text{ such that } B_k \in I \text{ for } 1 \leq k \leq n. \end{array} \right\}.$$

The third semantics using refutation is defined by

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

These three semantics are shown to be identical by Jaffar, et al.[9] .

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

$$PS(S) = \{A \in B(S) \mid \Gamma \vdash A\}.$$

Theorem 3.1 *For every EFS S,*

$$M(S) = lfp(T_S) = SS(S) = PS(S).$$

Proof $lfp(T_S)$ is identical to $T_S \uparrow \omega$, the set of all atoms in the Herbrand interpretations constructed by finite applications of T_S to empty set ϕ. Then clearly from definitions $T_S \uparrow \omega = \{A \in B(S) \mid \Gamma \vdash A\}$.

Thus the refutation is complete as a procedure of accepting EFS languages.

3.3 Negation as failure for EFS

Now we discuss the inference of negation. We prepair some definitions.

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

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

In the discussion of negation, we assume that any computation rule R makes all derivations *fair*. We say such a computation rule to be *fair*.

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

$$FF(S) = \left\{ A \in B(S) \;\middle|\; \begin{array}{l} \text{for any fair computation rule, there is an } n \text{ such that} \\ \text{all derivations from} \leftarrow A \text{ are finitely failed within length } n \end{array} \right\}.$$

Put $\overline{\theta} = (x_1 = \tau_1 \wedge \ldots \wedge x_n = \tau_n)$ for a substitution $\theta = \{x_1 := \tau_1, \ldots, x_n := \tau_n\}$, and for an empty θ, $\overline{\theta} = true$. By Jaffar, et al. [9], negation as failure for EFS is complete if the following two are satisfied:

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

(3.5) $FF(S)$ is the identical to the set

$$GF(S) = \left\{ A \in B(S) \;\middle|\; \begin{array}{l} \text{for any fair computation rule, all derivations} \\ \text{from} \leftarrow A \text{ are finitely failed} \end{array} \right\}.$$

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

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

$$GGF(S) = \left\{ A \in B(S) \;\middle|\; \begin{array}{l} \text{for any fair computation rule, all derivations} \\ \text{from} \leftarrow A \text{ such that all goals in them} \\ \text{are ground are finitely failed} \end{array} \right\}.$$

The inference rule that infers $\neg A$ for a ground atom A if A is not in $GGF(S)$ is called *Herbrand rule*[10].

Theorem 3.2 *For any variable-bounded EFS S,*

$$FF(S) = GF(S) = GGF(S).$$

Proof Since S is variable-bounded, every goal derived from a ground goal is ground, which means $GF(S) = GGF(S)$. Then from Proposition 3..1 the number of such goals is finite. Thus $FF(S) = GF(S)$ from König's Lemma.

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

$$E^* = \left\{ \tau = \pi \rightarrow \bigvee_{i=1}^{k} \overline{\theta_i} \;\middle|\; \begin{array}{l} \tau \text{ is a term, } \pi \text{ is a ground term,} \\ \text{and } \theta_1, \ldots, \theta_k \text{ are all unifiers of } \pi \text{ and } \tau \end{array} \right\}.$$

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

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 Σ be an alphabet with at least two symbols. Then a language $L \subset \Sigma^+$ 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 Σ^+. The converse is clear from Smullyan [17].

The left to right part of Theorem 4..1 is still valid in case alphabet Σ is singleton. However, to show the converse we need to weaken the statement slightly just as in Theorem 4..2(2) below.

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 Σ_0 of Σ, 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 Lemma 2..1.

(2) can also be proved by a simulation.

The Σ_0 above corresponds to the auxiliary alphabet like tape symbols or non-terminal symbols. We can show another theorem related to the converse of Theorem 4..2(1).

Definition A function σ from Σ^+ 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 $\Gamma_0 \vdash p(u, w) \Leftrightarrow w = \sigma(u)$.

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

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

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 Π consists of unary predicate symbols and for each axiom in Γ 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 $\Gamma = \{p(a) \leftarrow, \ p(xx) \leftarrow p(x)\}$ is simple and $L(S, p) = \{a^{2^n} \mid n \geq 0\}$.

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

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

Example 4.2 An EFS $S = (\{a, b\}, \{p\}, \Gamma)$ with $\Gamma = \{p(ab) \leftarrow, \ p(axb) \leftarrow p(x)\}$ 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 Γ is one of the following forms:

$$\begin{array}{ll} p(\pi) \leftarrow, & \\ p(ux) \leftarrow q(x) & \qquad (p(ux) \leftarrow q(x)), \end{array}$$

where π 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 definable if and only if it is regular.*

The pattern languages [1, 2, 14, 15] 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 π 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 π contains at most one variable.*

Proposition 4.2 (Shinohara [14]) *Let w be a word in Σ^+ and π be a regular pattern. Then each unifier of w and π is computed in $O(|w| + |\pi|)$ time.*

By these propositions, the unifier of w and π 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 5 items, the set of rules, the representation of rules, the data presentation, the method of inference called *inference machine*, and the criterion of successful inference [4].

In our problem, the class of rules are EFS languages. The examples are ground atoms $p(w)$ with sign $+$ or $-$ indicating whether $p(w)$ is provable from the target EFS or not. An example $+p(w)$ is said to be *positive*, $-p(w)$ *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) [13]. 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 $p(w)$ for some negative example $-p(w)$. H is *too weak*, if H can not prove $p(w)$ for some positive example $+p(w)$.

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 generator and show its completeness.

5.1 Contradiction backtracing algorithm for EFS

Contradiction backtracing algorithm (CBA for short) devised by Shapiro[13] 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_i 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 $G$ false in $M$}
output:     a clause $C_i$ false in $M$;
   begin
      for $i := k$ downto 1 do begin
         let $A_i$ be the selected atom of $G_{i-1}$;
         if $ASK(A_i)$ is false then return $C_{i-1}$;
      end
   end
```

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 θ and A be the selected atom of G. Assume 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 $G = \leftarrow \Psi_1, A, \Psi_2$ be a ground goal and $C = A' \leftarrow \Psi_3$ be a variable-bounded clause, where Ψ_1, Ψ_2 and Ψ_3 are sequences of atoms, and A and A' are atoms unifiable by a substitution θ. Then $G' = \leftarrow \Psi_1, \Psi_3\theta, \Psi_2$ is a ground resolvent of G and C. Since we assume G' is false in a model M, all atoms in Ψ_1, Ψ_2 and $\Psi_3\theta$ are ground and true in M. Therefore if A is true in M, then $G = \leftarrow \Psi_1, A, \Psi_2$ is false in M, otherwise $C\theta = A \leftarrow \Psi_3\theta$ is false in M.

Theorem 5.1 *Let M be a model of a variable-bounded EFS S, and $(G_0 = G, \theta_0, C_0)$, (G_1, θ_1, C_1), ..., $(G_k = \Box, C_k, \theta_k)$ be a refutation by S of a ground goal G false in M. If CBA is given the refutation, then it makes i oracle calls and returns C_{i-1} false in M for some $i = 1, 2, \ldots, k$.*

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 G_0 is not empty. Hence $k - i$ is positive. If CBA makes the k-th call to the oracle ASK, then received truth value of A_1 upon which G_1 is resolved must be false because A_1 is identical to an atom in G_0. Therefore CBA always returns a clause C_{i-1} after making at most k oracle calls.

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 size$(\pi) = 2 \times |\pi| - |v(\pi)|$ for a pattern π, and size(C) = size$(\pi\tau_1 \ldots \tau_n)$ for a clause $C = p(\pi) \leftarrow q_1(\tau_1), \ldots, q_n(\tau_n)$.

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

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

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

(5.1) $\theta = \{x := y\}$, where $x \in v(C)$, $y \in v(C)$ and $x \neq y$,

(5.2) $\theta = \{x := a\}$, where $x \in v(C)$ and $a \in \Sigma$, or

(5.3) $\theta = \{x := yz\}$, where $x \in v(C)$, $y \notin v(C)$, $z \notin v(C)$ and $y \neq z$.

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

Proof If θ 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 θ is of the form $\{x := yz\}$, then $|C\theta| = |C| + o(x, C)$ and $|v(C\theta)| = |v(C)| + 1$. Since $o(x, C) \geq 1$,

$$\text{size}(C\theta) = \text{size}(C) + 2 \times o(x, C) - 1 > \text{size}(C).$$

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

(5.4) $A = \Box$ and $B = p(x)$ for $p \in \Pi$, $x \in X$, or

(5.5) $A\theta = B$ for a substitution θ basic for A.

Lemma 5.3 *Let C and D be clauses such that $C\theta = D$ but $C \not\equiv D$ for some substitution θ. Then there exists a sequence of substitutions θ_1, θ_2, ..., θ_n such that θ_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 *ρ_a is a locally finite and complete refinement operator for atoms.*

Shinohara [14] discussed inductive inference of pattern languages from positive data. The method he called *tree search method* uses a special version of the refinement operator ρ_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 $\rho_{vb}(C)$ if and only if (5.4) or (5.5) holds, or

$$C = A \leftarrow B_1, \ldots, B_{n-1} \text{ and } D = A \leftarrow B_1, \ldots, B_{n-1}, B_n \text{ is variable-bounded.}$$

Similarly we define ρ_{lb} for length-bounded clauses.

Theorem 5.3 *ρ_{vb} is a complete refinement operator for variable-bounded clauses.*

Theorem 5.4 *ρ_{lb} is a locally finite and complete refinement operator for length-bounded clauses.*

Note that ρ_{vb} is not locally finite because the number of atoms B_n possibly added by ρ_{vb} is infinite, while ρ_{lb} is locally finite. We can also define refinement operators for simple or regular clauses and prove they are locally finite 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 the 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 [20, 12] 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, 18], and introduction of the empty word [15] which we will discuss elsewhere.

References

[1] D. Angluin. Finding patterns common to a set of strings. In *Proc. 11th Annual ACM Symp. Theory of Computing*, 130–141, 1979.

[2] D. Angluin. Inductive inference of formal languages from positive data. *Inform. Contr.*, 45:117–135, 1980.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Inform. Comp.*, 75:87–106, 1987.

[4] D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15:237–269, 1983.

[5] S. Arikawa. Elementary formal systems and formal languages - simple formal systems. *Memoirs of Fac. Sci., Kyushu Univ. Ser. A.*, Math. 24:47–75, 1970.

[6] M. Fitting. *Computability Theory, Semantics, and Logic Programming.* Oxford Univ. Press, 1987.

[7] E. M. Gold. Language identification in the limit. *Inform. Contr.*, 10:447–474, 1967.

[8] H. Ishizaka. Inductive inference of regular languages based on model inference. *To appear in IJCM*, 1989.

[9] J. Jaffar, J.-L. Lassez, and M. J. Mahr. Logic programming scheme. In D. DeGroot and G. Lindstrom, editors, *Logic Programming: Functions, Relations, and Equations*, 211–233, 1986.

[10] J. W. Lloyd. *Foundations of Logic Programming: Second, Extended Edition.* Springer - Verlag, 1987.

[11] G. D. Plotkin. Building in equational theories. In *Machine Interigence 7*, 132–147, 1972.

[12] Y. Sakakibara. Learning context-free grammars from structural data in polynomial time. In *Proc. COLT'88*, 296–310, 1988.

[13] E. Y. Shapiro. Inductive inference of theories from facts. Research Report 192, Yale Univ., 1981.

[14] T. Shinohara. Polynomial time inference of pattern languages and its application. In *Proc. 7th IBM Symp. Math. Found. Comp. Sci.*, 191–209, 1982.

[15] T. Shinohara. Polynomial time inference of extended regular pattern languages. *LNCS*, 147:115–127, 1983.

[16] T. Shinohara. Inductive inference of formal systems from positive data. *Bull. Inform. Cyber.*, 22:9–18, 1986.

[17] R. M. Smullyan. *Theory of Formal Systems.* Princeton Univ. Press, 1961.

[18] L. G. Valiant. A theory of the learnable. *CACM*, 27(11):1134–1142, November 1984.

[19] A. Yamamoto. A theoretical combination of SLD-resolution and narrowing. In *Proc. 4th ICLP*, 470–487, 1987.

[20] T. Yokomori. Learning simple languages in polynomial time. In *SIG–FAI JSAI*, 21–30, 1988.