TR-217

Inductive Inference of Context-free Languages —Context-free Expression Method—

by

T. YOKOMORI

November, 1986

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Inductive Inference of Context-free Languages
— Context-free Expression Method —

by

Takashi YOKOMORI*

*)Research Staff, Fundamental Informatics Section, International Institute for Advanced Study of Social Information Science, FUJITSU LIMITED.
140 Miyamoto, Numazu, Shizuoka 410-03 JAPAN

Abstract

An inductive inferece problem of context-free languages is considered. There have been many attempts to this problem, and most of them are based on a problem setup in which a representation space for hypotheses is a class of context-free grammars. An inference algorithm given in this paper , on the contrary, employs a kind of extensions of regular expressions called context-free expressions as a representation space for context-free languages. The algorithm, based on the notion of an identification in the limit, is significantly concise when compared with existing algorithms.

Then, a subclass of context-free languages is examined, and a simpler algorithm and its improved version for the subclass are presented together with some complexity results on the inference problems.

Further, a topic on meta inductive inference is briefly discussed.

1. Introduction

An inductive inference is, in general, recognized as a process of finding a set of rules from given many examples. The mechanism underlying is one of the most significant functions for supporting knowledge acquisition process in the various phases of problem solving we encounter, and it is also one of the primary subjects in the research on machine learning.

For the present, the research efforts in the inductive inference problem mainly forcus on the domains of finite-state automata, formal grammars, first-order logic, LISP programs, and so on, and some results have been obtained in its own domain. There remain , however, many to be done in the context of developing practical inference algorithm for solving realistic application issues such as a problem of automatic programming from examples.

Now, we consider the following model of inductive inference problem: Given an object L of inference, an inductive inference device (IID) tries to infer a representation H for the object from examples. It is assumed that IID has an enumeration mechanism by which any possible hypothesis from the representation space can be eventually enumerated at least once. It is also assumed that we can utilize an oracle concerning examples from the object. IID asks the oracle for an example, and computes hypothesis and outputs it, and again asks another example for the next step, and this process is cycled. In a sequence of hypotheses H_1, H_2, ... IID is said to identify L in the limit if there exists a positive integer n such that H_n represents L and H_{n+i} equals to H_n for all $i>0$.

A simple algorithm for identification in the limit is the one based on the notion of identification by enumeration. Let H_1, H_2, ... be an effective enumeration of the possible hypotheses, and suppose a set of examples $e_1, e_2, ..., e_k$ are presented. Then, IID provides as its next output the first hypothesis which is compatible with all these examples. Under the assumption of a perfect oracle, the sequence of hypotheses converges in the limit.

Now, Shapiro([Sh 82]) extends this idea to the domain of first-order logic, and constructs a program which infers sentences from examples of their logical concequences, which is recognized as the first attempt and fruit(success) in the area. His representation space for hypotheses is restricted to the class of sentences in clausal forms, and the alphabet of the representation language is fixed to be finite. The object of the inference in his strategy is the Herbrand model , and examples from the model are given in the form of ground atoms. Although Shapiro's method has really a fruitful success, as Laird suggested[La86], it is not necessary true that the first-order logic is best suitable for general purpose as a representation space for hypotheses. For example, it is indeed possible to represent regular sets in terms of Horn logic, however, regular expressions or finite-state transition graphs can provide a better and more intuitive way of representing regular sets.

This is exactly the piont which motivated this work. In the next section, we consider the inductive inference problem for context-free languages, and employ a representation space for hypotheses different from the ones in the exsisting methods. This enables us to make an elegant discussion on the problem and the algorithm for solving the problem.

This paper is organized as follows: Fisrt, a formal framework for discussing the problem of inductive inference is given in Section 2. Then, Section 3 deals with the inductive inference problem for context-free languages and gives an inference algorithm for the problem, in which the notion of a context-free expression plays a central role. In Section 4, a subclass of context-free languages called semilinear languages is investigated as the inference object, and simpler and improved algorithms for inferring the class are presented, together with some complexity results. In Section 5, a topic on meta inference is briefly mentioned, and followed by a concluding remarks in Section 6.

2. Formal Framework for Inductive Inference

There have been a number of attempts to formalize a problem of inductive inference. Among them, the concept of an identification in the limit devised by Gold takes a central position in the framework for inductive inference of formal languages. Here we adopt a formal definition of more abstract problem setup which has been recently proposed by Laird.

An abstract inference problem is defined as follows.

Definition 2.1([La86])

An *abstract inference problem* is a 6-tuple (D, d_0, Ω, h, ASK, EX), where

(1) D is a finite or countable set partially ordered by $\geqq$ (called *semantic domain of objects*),

(2) d_0 is a designated element of D (called *target*),

(3) Ω is a countable set of expressions (called *representation space*),

(4) h: $\Omega \rightarrow$ D is a surjective mapping from expressions to objects,

(5) ASK is an oracle for $\geqq$ such that ASK(e_1,e_2)=1 if $h(e_1) \geqq h(e_2)$, and =0 otherwise($e_1, e_2 \in \Omega$),

(6) EX is an oracle for examples of d_0 such that if EX() = +e then $d_0 \geqq h(e)$, and if EX() = −e then $d_0 \ngeqq h(e)$(e is an example of d_0).

Note. The example set of d_0 consists of both positive (signed +) and negative (signed−) examples with respect to d_0. EX() has no input, denoted by (), and produces +e or −e as an output per each call of the oracle.

This formalization provides a simple formal framework for the inductive inference problem setup in which each example on the target can be regarded as an element of representation space. In fact, using this framework it is possible to deal with several existing inference problems with representaion spaces such as logic programs, regular expressions.

Now, it is well recognized that the way of presenting examples plays a significant role in the inductive inference problem([Go67]). The following definition is often used in almost all the work reported so far.

Definition 2.2([La86])

The oracle EX () is said to give a *complete presentation of d_0* if for every $e \in \Omega$ such that $d_0 \geqq h(e)$, EX() eventually returns "$+e$" at least once, and for every $e \in \Omega$ such that $d_0 \ngeqq h(e)$, EX() eventually returns "$-e$" at least once.

Then, it is known that an inference algorithm for the abstract inference problem is given as follows :

Algorithm A_0 (Identification by enumeration [La86])

```
Input :        A recursively enumerable set of $\Omega$ of expressions
               An oracle ASK($e_1$,$e_2$) for whether $h(e_1) \geqq h(e_2)$ or not?
               An oracle EX() for a complete presentation of $d_0$
Output :       A sequence of expressions $e_1$, $e_2$, ... such that $e_n$ is correct for the
               first n examples
Procedure :    Let $e_1$, $e_2$,... be an enumeration of $\Omega$
               $i \leftarrow 1$
               EXAMPLES $\leftarrow \Phi$ (empty set)
               do forever :
                   EXAMPLES $\leftarrow$ EXAMPLES $\cup$ EX() (get next example)
                   while ASK($e_i$,e) = 1 for some negative example $-e$ or ASK($e_i$,e)
                   = 0 for some positive example $+e$
                       $i \leftarrow i+1$
                   Output $e_i$ as the next hypothesis.
```

The correctness of Algorithm A_0 is guaranteed by the following .

Theorem 2.1 ([La86])

Algorithm A_0 *identifies* d_0 *in the limit.*

Turning to the inductive inference problem for formal languages in general, it is very common for the problem setup to take formal grammars as its representation space. And , several results on the inference problem of regular sets have been successfully reported, while only a little is known about the effective, complete algorithm for the inference problem of the language classes larger than regular sets ([An78],[An86],[Bi72],[Go78],[ET76],[Shi83],[TA77],[Wh77],[Ta86]).

In this paper, we present an inductive inference algorithm for the class of context-free languages. The abstract inference problem given above is employed as a formal framework of the problem. Hence, it is necessary to set up the problem so that each example on the target may be regarded as an element of the representation space.

Our approach is unique in that it does not employ the class of context-free grammars as a representation space. Instead, we use a kind of extension of regular expressions called "context-free expressions". This makes it possible to formalize the inference problem of context-free languages as an instance of the abstract inference problem and also to provide a very simple algorithm for the problem.

3. Inductive Inference of Context-free Languages

In this section, the notion of a context-free expression is introduced which constitutes the representation space for context-free languages. The context-free expression, which has been originally proposed by Gruska ([Gr71]), is a representation form for a context-free language and is a natural extension of a regular expression.

3.1 Context-free Expressions——Extended Regular Expressions

We shall give some basic notions and notations needed through this paper. (The reader is assumed to be familiar with the rudiments in the formal language theory. See, e.g., [Sa73] for definitions unstated here.)

For a given finite alphabet , the set of all strings with finite length (including zero) is denoted by Σ^*. (An empty string is denoted by ε.) A *language* L *over* Σ is a subset of Σ^*. For an infinite alphabet Γ, L is a language over Γ if L is a language over some finite subset Σ of Γ.

A *product* of languages L_1 and L_2, denoted by L_1L_2, is defined as follows: $L_1L_2=\{xy \mid x \in L_1, y \in L_2\}$. For a language L and an integer $i \geq 0$, L^i is defined inductively : $L^0=\{\varepsilon\}$, $L^{i+1}=L^iL$. Further, we define $L^*=\cup_{i\geq 0}L^i$, and $L^+=\cup_{i\geq 1}L^i$, and call **-closure* and *+-closure* of L, respectively.

A *context-free grammar* is a 4-tuple G=(N,T,P,S), where N is a finite alphabet of nonterminals, T is a finte alphabet of terminals such that $N \cap T=\Phi$, S is a distinguished element of N called the initial symbol, and P is a finite set of production rules of the form $A \rightarrow w$ ($A \in N$, $w \in (N \cup T)^*$). For x, y $\in (N \cup T)^*$, a binary relation $\Rightarrow$ is defined as follows: $x \Rightarrow y$ iff there exist u, $v \in (N \cup T)^*$, $A \rightarrow w \in P$ such that $x=uAv$ and $y=uwv$. Let $\Rightarrow^*$ be the transitive, reflexive closure of $\Rightarrow$. A set $L(G)=\{x \mid S \Rightarrow^* x$ and $x \in T^*\}$ is called the language generated by G. A language is called *context-free* if there exists a context-free grammar G such that L=L(G) holds.

Now, the following operation plays a crucial role in this paper.

Definition 3.1 ([Gr71])

(i) Let σ be a symbol and L_1, L_2 be languages. Then, *σ-substitution of L_2 into L_1* , denoted by $L_1 \uparrow^{\sigma} L_2$, is defined as follows:

$L_1 \uparrow^{\sigma} L_2 = \{x_1y_1 \cdots x_ky_kx_{k+1} \mid x_1\sigma \cdots x_k\sigma x_{k+1} \in L_1$, σ does not occur in the word

$x_1 \cdots x_{k+1}$ and $y_i \in L_2$ for $1 \leq i \leq k\}$

(ii) Let σ be a symbol and L be a language. Then, *σ-iteration of L*, denoted by L^{σ}, is defined by

$$L^{\sigma}=\{x \mid x \in L \cup L \uparrow^{\sigma} L \cup L \uparrow^{\sigma} L \uparrow^{\sigma} L \cup \cdots, \text{and } x \text{ has no occurrence of } \sigma\}.$$

Remarks.

(1)If L_1 does not contain σ, then $L_1 \uparrow^{\sigma} L_2 = L_1$.

(2)Let L be a language over T and suppose that T does not contain σ. Then, $L^*=(L\sigma \cup \{\varepsilon\})^{\sigma}$ and $L^+=(L\sigma \cup L)^{\sigma}$.

Now, we introduce a notation for representing context-free languages, called "context-free expression".

Definition 3.2([Gr71])

Let Γ be a (possibly infinite) alphabet, Γ' be the boldface version of Γ, i.e, $\Gamma' = \{\boldsymbol{\sigma} | \sigma \in \Gamma\}$. Then, *context-free expressions over* Γ are strings over $\Gamma \cup \Gamma' \cup \{\phi, +, (,)\}$ defined as follows:

(i) ϕ is a context-free expression,

(ii) if a is in $\Gamma \cup \{\varepsilon\}$, then a is a context-free expression,

(iii) if E_1, E_2 are context-free expressions and $\sigma \in \Gamma$, then $(E_1 + E_2)$, E_1E_2, $(E_1)\boldsymbol{\sigma}$ are context-free expressions,

(iv) nothing else is a context-free expression.

For each context-free expression over Γ, a language over Γ is associated in the following convention:

Definition 3.3([Gr71])

A mapping $|\,|$ from context-free expressions to a class of languages is defined by:

(i) $|\phi| = \Phi$

(ii) $|a| = \{a\}$ (for $\forall a \in \Gamma \cup \{\varepsilon\}$)

(iii) $|E_1 + E_2| = |E_1| \cup |E_2|$, $|E_1E_2| = |E_1||E_2|$, and $|E\boldsymbol{\sigma}| = |E|^{\sigma}$.

Example 3.1

Let a,b, σ be in Γ, then $E = (a\sigma b + ab)\boldsymbol{\sigma}$ is a context-free expression. Further, $|E| = |(a\sigma b + ab)\boldsymbol{\sigma}| = \{a\sigma b, ab\}^{\sigma} = \{a^i b^i | i \geqq 1\}$.□

Definition 3.4 (Language Class Ω_Γ)

Let be Γ a (possibly infinite) alphabet. Then, a class of languages Ω_Γ is defined as follows:

(1) $\Phi, \{\varepsilon\} \in \Omega_\Gamma$,

(2) if $a \in \Gamma$, then $\{a\} \in \Omega_\Gamma$,

(3) if $L_1, L_2 \in \Omega_\Gamma$ and $\sigma \in \Gamma$, then $L_1 \cup L_2$, L_1L_2, and $L_1{}^{\sigma} \in \Omega_\Gamma$.

Now, the next result plays an important role in this paper.

Theorem 3.1 ([Gr71])

Let Σ *be a finite alphabet, and let* L *be a language over* Σ. *Then,* L *is a context-free language if and only if there exists a finite alphabet* T *such that* $\Sigma \subseteq T$ *and* $L \in \Omega_T$.

Therefore, we have the following characterization of context-free languages in terms of context-free expressions.

Theorem 3.2

Let Σ *be a finite alphabet, and let* L *be a language over* Σ. *Then, L is a context-free language if and only if there exists a finite alphabet* T *and a context-free expression* E *over* T *such that* $\Sigma \subseteq T$ *and* $|E| = L$.

Proof. Obvious from Theorem 3.1 and definitions.□

Thus, context-free expressions provide a way of representing context-free languages, and for our purpose , i.e., for developing an algorithm of the inductive inference problem for context-free languages, it can provide a better representation space than any other device like grammars. That is, as is shown later, an inductive inference algorithm for context-free languages is obtained by naturally extending the one for regular sets in terms of regular expressions.

3.2 Inductive Inference Algorithm

In this paper, we formalize the inductive inference problem for context-free languages as follows:

<Inductive Inference Problem for Context-free Lanugages>

(1) a semantic domain D is the class of context-free languages, and a partial order $\supseteq$ is an inclusion relation over D,

(2) d_0 is a given context-free language,

(3) a representation space Ω is the class of contex-free expressions,

(4) for E in Ω , h(E) is defined as a language $|E|$ in D,

(5) an oracle EX() works as follows:

EX() = +e implies $h(e) \subseteq d_0$, and EX() = −e implies $h(e) \not\subseteq d_0$.

Now, let L be a context-free language over a finite alphabet Σ. We fix an infinite alphabet Γ over which context-free expressions are defined, where $\Sigma \subseteq \Gamma$.(It is assumed that auxiliary symbols ϕ, +, (,) are not contained in Γ.)

We define an operator δ on Ω, the set of all context-free expressions over Γ, as follows:

Suppose E, E_1, E_2 are context-free expressions over Γ. As a notation, by $E_1 \rightarrow E_2$ we denote $E_2 \in \delta(E_1)$:

(1) $\phi \rightarrow \phi\phi$

(2) $\phi \rightarrow a \ (\forall a \in \Gamma \cup \{\varepsilon\})$

(3) $\phi \rightarrow (\phi)\sigma \ (\forall \sigma \in \Gamma)$

(4) $\phi \rightarrow (\phi + \phi)$

(5) if $E_1 \rightarrow E$, then $E_1 + E_2 \rightarrow E + E_2$ and $E_2 + E_1 \rightarrow E_2 + E$

(6) if $E_1 \rightarrow E$, then $E_1\sigma \rightarrow E\sigma \ (\forall \sigma \in \Gamma)$

(7) if $E_1 \rightarrow E$, then $E_1E_2 \rightarrow EE_2$ and $E_2E_1 \rightarrow E_2E$.

Note that there are two types of rules forming the operator δ: one for rewriting ϕ ((1)-(4)), the other for preserving structure of expressions((5)-(7)).

Lemma 3.1

The operator δ defined above has the following properties:

(i) *δ is complete for the most specific expression ϕ in the sense that the set $\delta^*(\phi)$ of all expressions obtainable from ϕ in a finite number of applications of δ includes at least one expression for every context-free language.*

(ii) *For arbitrary expressions E_1, E_2 in Ω, whenever $E_1 \in \delta(E_2)$, $|E_1| \supseteq |E_2|$ holds.*

Proof.

We prove by induction on the way of constructing expressions in Definition 3.2.

(i) By Theorem 3.2, it suffices to show that $\Omega \subseteq \delta^*(\phi)$. First, by the rewriting rule (2) above, we have that $a \in \delta(\phi)$ for $\forall a \in \Gamma \cup \{\varepsilon\}$. Now, suppose that E_1 and E_2 are in $\delta^*(\phi)$. (In what follows, by $\rightarrow^*$ we denote a finite number of applications of $\rightarrow$. Then,

$\phi \to \phi+\phi$ (by (4)) $\to^*$ $E_1+\phi$ (by applying (5) together with the induction hypothesis)$\to^* E_1+E_2$ (by applying (5) together with the induction hypothesis), thus, we have $E_1+E_2 \in \delta^*(\phi)$. Similarly,

$\phi \to \phi\phi$ (by (1)) $\to^*$ $E_1\phi$ (by applying (7) together with the induction hypothesis) $\to^* E_1E_2$ (by applying (7) together with the induction hypothesis), hence we have $E_1E_2 \in \delta^*(\phi)$. Further, $\phi \to \phi\,\sigma$ (by (3)) and since $\phi \to^* E_1$, by applying (6) repeatedly, we have $\phi\,\sigma \to^* E_1\sigma$, hence $\phi \to^* E_1\sigma$, i.e. $E_1\sigma \in \delta^*(\phi)$.

(ii) Since, from the rules (1)-(4), $\delta(\phi)=\{\phi\phi, \varepsilon, \phi+\phi, \sigma, \phi\sigma(\forall\sigma\in\Gamma)\}$, we have that for each $E \in \delta(\phi)$, $|E| \supseteq |\phi|=\Phi$.

Now, let E' be in $\delta(E)$. Then, there are only three cases concerning E.(Note that if E is in $\Gamma\cup\{\varepsilon\}$, then no rule in δ is applicable to E.)

① $E=E_1+E_2$; Let $E_i' \in \delta^*(E_i)(i=1,2)$. Then, by the induction hypothesis, $|E_i'| \supseteq |E_i|$ holds.

Hence, by (5) $|E|=|E_1+E_2|=|E_1|\cup|E_2| \subseteq |E_1'|\cup|E_2|=|E'|$ or

$$|E|=|E_1+E_2|=|E_1|\cup|E_2| \subseteq |E_1|\cup|E_2'|=|E'|$$

is obtained.

② $E=E_1E_2$; By the same hypothesis,

$$|E|=|E_1E_2|=|E_1|\,|E_2| \subseteq |E_1'|\,|E_2|=|E'| \text{ or}$$

$$|E|=|E_1E_2|=|E_1|\,|E_2| \subseteq |E_1|\,|E_2'|=|E'|$$

is obtained.

③$E=E_1\sigma(\forall\sigma\in\Gamma)$; Let $E_1' \in \delta^*(E_1)$. Then, by the induction hypothesis, $|E_1'| \supseteq |E_1|$ holds. Hence by (6)

$$|E|=|E_1\sigma|=|E_1|\,\sigma \subseteq |E_1'|\,\sigma=|E'|$$

is obtained. This completes the proof.□

Example 3.2 (Derivation process of expressions)

Consider the following derivation process from ϕ :

	key rule used
$\phi \to (\phi)_\tau$	(3)
$\to (\phi\phi)_\tau$	(1)

$\rightarrow ((\phi+\phi)\ \phi)\tau$ (4)

$\rightarrow ((\phi\phi+\phi)\ \phi)\tau$ (1)

$\rightarrow ((a\phi+\phi)\ \phi)\tau$ (2)

$\rightarrow ((a(\phi)\sigma+\phi)\ \phi)\tau$ (3)

$\rightarrow ((a(\phi+\phi)\sigma+\phi)\ \phi)\tau$ (4)

$\rightarrow^* ((a(\phi\phi\phi+\phi)\sigma+\phi)\ \phi)\tau$ (1)

$\rightarrow^* ((a(a\sigma\sigma+b)\sigma+\phi)\ \phi)\tau$ (2)

$\rightarrow ((a(a\sigma\sigma+b)\sigma+\phi\phi)\ \phi)\tau$ (1)

$\rightarrow ((a(a\sigma\sigma+b)\sigma+b\phi)\ \phi)\tau$ (2)

$\rightarrow ((a(a\sigma\sigma+b)\sigma+b(\phi)v)\ \phi)\tau$ (3)

$\rightarrow ((a(a\sigma\sigma+b)\sigma+b(\phi+\phi)v)\ \phi)\tau$ (4)

$\rightarrow^* ((a(a\sigma\sigma+b)\sigma+b(\phi\phi\phi+\phi)v)\ \phi)\tau$ (1)

$\rightarrow^* ((a(a\sigma\sigma+b)\sigma+b(bvv+a)v)\ \phi)\tau$ (2)

$\rightarrow ((a(a\sigma\sigma+b)\sigma+b(bvv+a)v)\ (\phi+\phi))\tau$ (4)

$\rightarrow {}^*((a(a\sigma\sigma+b)\sigma+b(bvv+a)v)\ (\tau+\varepsilon))\tau$ (2)

Thus, we have eventually obtain an expression :

$E=((a(a\sigma\sigma+b)\sigma+b(bvv+a)v)(\tau+\varepsilon))\tau$

and its language is :

$|E|=\{w|\ \#_a(w)=\#_b(w),\ w\in\{a,b\}^*\}$, where $\#_x(w)$ denotes the number of a letter x appearing in w.

(By the way, the language $|E|$ is generated by a grammar with the initial symbol S and the set of rules : $\{S\rightarrow aBS,\ S\rightarrow aB,\ S\rightarrow bAS,\ S\rightarrow bA,\ A\rightarrow bAA,\ A\rightarrow a,\ B\rightarrow aBB,\ B\rightarrow b\}$.)□

Now, using the operator δ defined above, we can obtain an algorithm for the inductive inference problem of context-free languages, which is quite simple and based on the principle of "identification by enumeration".

We assume the oracle EX for the set of positive and negative examples. Note that since a positive example e can be regarded as an expression E whose language $|E|$ is $\{e\}$, the set of positive examples constitutes a subset of the representation space Ω.

Definition 3.5

(i) (complete and sufficient oracle)

Let d_0 be a context-free language of target. An oracle EX is called *complete* and *sufficient for d_0* if there exists a signed subset K of Ω satisfying the followings :

(1) the set $\{E \in \Omega |$ for all e in K, if e is positive, then $h(e) \subseteq h(E)$, else $h(e) \nsubseteq h(E)\}$ is exactly the set $\{E \in \Omega | \ h(E) = d_0\}$.

(2) for every $e \in K$ such that $d_0 \supseteq h(e)$, EX() eventually returns "$+e$" at least once, and for every $e \in K$ such that $d_0 \nsupseteq h(e)$, EX() eventually returns "$-e$" at least once,

(ii) (admissible presentation of d_0)

A presentation of examples of d_0 is called *admissible* if it has an oracle EX which is complete and sufficient for d_0.

It should be noted that there exists such a K for our case, that is, if we take K as the set $\{E \in \Omega \mid |E|$ is a singleton , and if it is in d_0, then E has a sign $+$, otherwise it has a sign $-\}$, then K satisfies the conditions mentioned above. Hence, we may assume the existence of the admissible presentation of d_0.

Before presenting an algorithm, we need some preliminaries.

For a given target context-free language L, let $T = \{a_1, \ldots, a_n\}$ be the alphabet over which L is defined. Further, for each $k \geqq 1$, let $\Gamma_k = T \cup \Delta_k \cup \Delta'_k \cup \{\phi, +, (,)\}$, where $\Delta_k = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, $\Delta'_k = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$.

Now, Theorem 3.2 says that for any context-free language L, there exist some integer $k \geqq 1$ and an expression E such that $|E| = L$ and E is a string over Γ_k. (Note that an integer k, in general, depends on L.)

[The outline of the algorithm]

We outline the inductive inference algorithm for context-free languages. For a given expression E and $i \geqq 0$, $k \geqq 1$, let $\delta(E,i,k) = \{E' | E \rightarrow^{i} E'$ and E' is a string over Γ_k $\}$.

Then, $\delta^*(E)=\cup_{i\geq 0}\cup_{k\geq 1}\delta(E,i,k)$. We abbreviate $\delta(\phi,i,k)$ as $\delta(i,k)$. The algorithm requires an enumeration procedure which, starting with ϕ, enumerates every expression. The enumeration is performed in the order indicated in Figure 1.

$$\begin{tikzcd}
\phi \arrow[r] & \delta(1,1) \arrow[r, Rightarrow] & \delta(2,1) \arrow[r, Rightarrow] \arrow[dl] & \delta(3,1) \arrow[r, Rightarrow] \arrow[dl] & \delta(4,1) \arrow[r, Rightarrow] \arrow[dl] & \cdots \\
(\phi \to) & \delta(1,2) \arrow[r, Rightarrow] \arrow[urr] & \delta(2,2) \arrow[r, Rightarrow] \arrow[dl] & \delta(3,2) \arrow[r, Rightarrow] \arrow[dl] & \cdots & \\
(\phi \to) & \delta(1,3) \arrow[r, Rightarrow] \arrow[uurrr] & \delta(2,3) \arrow[r, Rightarrow] \arrow[dl] & \cdots & & \\
(\phi \to) & \delta(1,4) \arrow[r, Rightarrow] \arrow[uuurrrr] & \cdots & & & \\
 & \vdots & & & &
\end{tikzcd}$$

Figure 1. The Enumeration Order

As a notation, we denote an element of $\delta(i,k)$ by $[E,(i,k)]$. Note that for each i,k, $\delta(i,k)$ is finite. Further, let $\delta(i+1,k)=\{E'|E\rightarrow E'$ and $E\in\delta(i,k)$, $E'\in\Gamma_k{}^*\}$, that is, an element of $\delta(i+1,k)$ is obtained from an element of $\delta(i,k)$ by applying δ_k once, where δ_k is a restriction of δ to Γ_k. Finally, each $\delta(1,k)$ is obtained from ϕ by one application of δ_k .

Now, the algorithm works as follows: By applying δ_k to ϕ, it enumerates a hypothesis (an expression) E and stores into a queue Q in the form of $[E,(i,k)]$. then refines (generalizes) each hypothesis by examples. The enumeration is performed in the order indicated in Figure 1, that is

$\delta(1,1)$, $\delta(2,1)$, $\delta(1,2)$, $\delta(3,1)$, $\delta(2,2)$, $\delta(1,3)$, $\delta(4,1)$, . . .

For a hypothesis E, it does not cover some positive example(it is called "*too specific*"), then the algorithm generalizes E in some way. Otherwise, if its language |E| includes some negative example(it is called "*too general*"), then the algorithm simply discards it. This is repeated until a correct hypothesis is found.

The algorithm is given below as Algorithm A_1.

For the sake of helping one understand the process of enumerating hypotheses in the algorithm, Figure 2 illustrates how the contents of a queue Q changes during the enumeration.(The contents of Q is represented as a rectangle whose length may change as the time goes. The shadow portion of Q denoting $\delta(1,k)$ is created only when $\delta(k+1,1)$ is produced from $\delta(k,1)$, and it is placed before $\delta(k+1,1)$).

Remarks.

(1) For given integers $i,k \geqq 1$, $\delta(i,k)$ is obviously computable.

(2) For any E and an example w, it is decidable whether $w \in |E|$ or not, that is , the membership problem for the class of context-free languages is decidable.

Theorem 3.3

For any given context-free language d_0, *the algorithm* A_1 *identifies* d_0 *in the limit.*

Proof. We need to show the following two: Fisrt , the algorithm A_1 converges some hypothesis E, secondly, the hypothesis E is correct for the target d_0.

From the completeness property of δ ((i) of Lemma 3.1) and Theorem 3.2, there is a chain of generalization steps from ϕ : $\phi = E_0 \rightarrow E_1 \rightarrow \cdots \rightarrow E_n = E$ and $|E| = d_0$. Here we assume that n is as small as possible for the given d_0. Property (ii) of Lemma 3.1 together with the minimality of n ensures that for each i, $|E_i| \subset |E_n|$. Then, there are strings w_i such that $w_i \in |E|$ and $w_i \notin |E_i|$. Let E_n be in $\delta(n,k)$.

[Proof for correctness] Assume the algorithm converges to some expression E: that is , there exist $i,k \geq 0$ and E in $\delta(i,k)$ such that E is correct for not only all examples in EXAMPLES but also every example given in the future. (That is the definition of "convergence".) Hence, E is correct for d_0.

__Algorithm A$_1$__ (Inference Algorithm for Context-free Languages)

Input: A recursively enumerable set of context-free expressions Ω
An enumeration operator δ
An admissible presentation of a target language d_0

Output: A sequence of expressions $E_1, E_2, \ldots$ such that E_n is correst for the first n examples.

Procedure:

```
    Q←δ(1,1);{elements of δ(1,1) are stored in the queue Q}
    EXAMPLES←Φ (empty set)
    X←next(Q);{next removes the top element of Q}
    do (forever):
        EXAMPLES←EXAMPLES∪EX() {get next example}
        while (let X=[E_j,(i,k)] be the j-th element of δ(i,k) , then )
         ∃e∈EXAMPLES s.t. E_j is not correct for e
            if E_j is "too specific"
            then do
                    if i≧2
                    then  if k=1 and j=1
                          then append δ(1,i)δ(E_1,1,1) to the tail of Q;
                                X←next(Q);
                          else append δ(E_j,1,k) to the tail of Q;
                                X←next(Q);
                    else  append δ(E_j,1,k) to the tail of Q;
                       X←next(Q);
            else    (E_j is "too general")
                    discard E_j ;
                    X←next(Q);
        Output E_j as the next hypothesis.
```

where

```
    E is "too specific" :=
          if   ∃+e∈EXAMPLES s.t. e∉|E|, then return true
          else return  false

    E is "too general" :=
          if   ∃−e∈EXAMPLES s.t. e ∈|E|, then return true
          else return  false
```

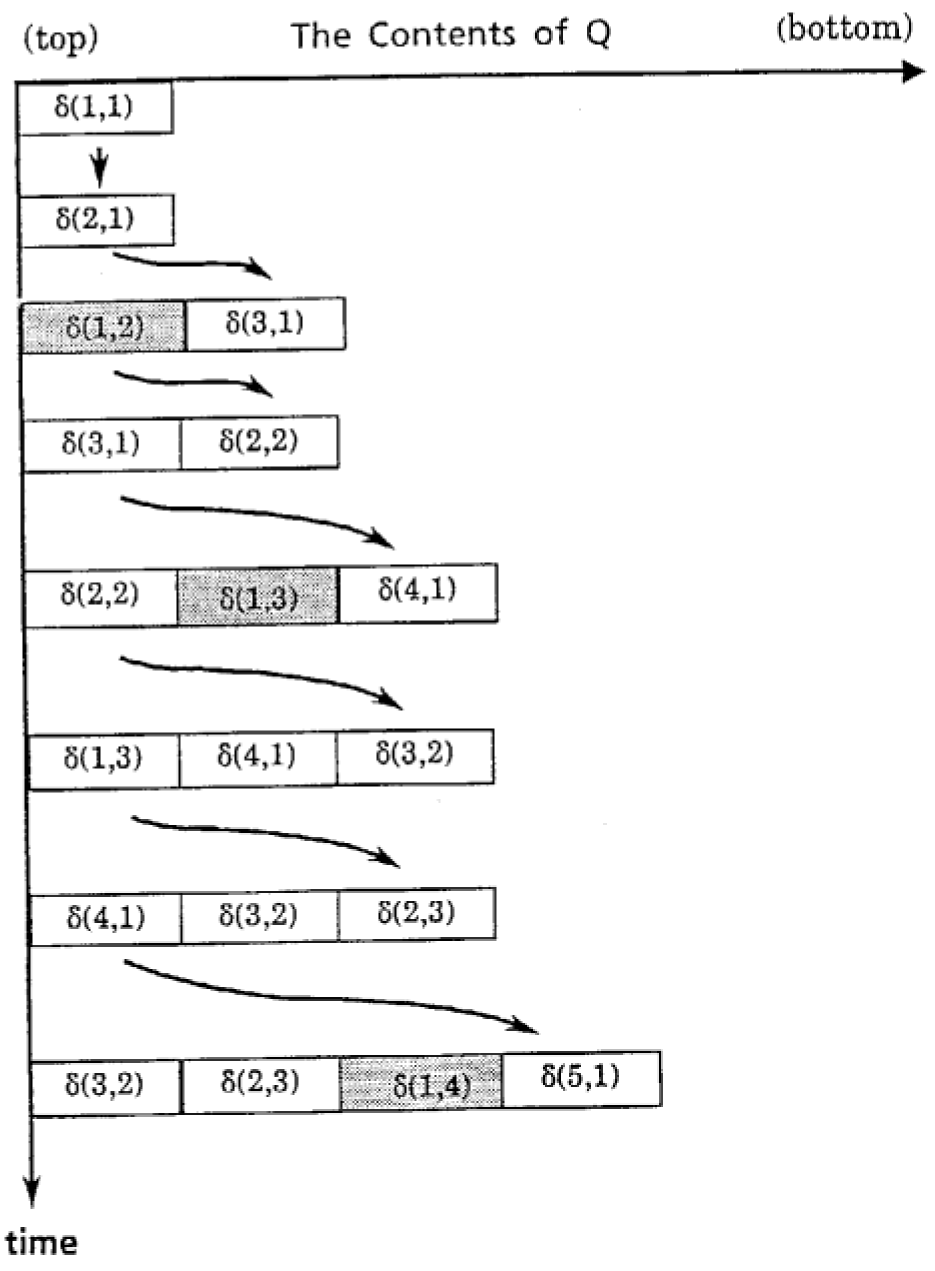

Figure 2. Queue Transition

[Proof for convergence] Suppose that the algorithm diverges. Then, we show, by the induction on j, that every hypothesis E_j $(0 \leq j \leq n)$in the chain above appears on the top of Q and is generalized. When $j=0$, it is trivial. Suppose that E_i appears on the top of Q as a hypothesis in $\delta(i,p)$ and is generalized. Since $E_{i+1} \in \delta(E_i)$, $E_{i+1} \in \delta(i+1,q)$ for some q. The divergence of the algorithm implies that the finite number of expressions preceding E_{i+1} will all appear on the top and be generalized or simply discarded. Hence, E_{i+1} eventually appears on the top of Q and is generalized due to w_{i+1}. Thus, $E_n (j=n)$ eventually appears on the top. However, because of the assumption of the divergence, E_n is not correct for some w, which contradicts the fact that $|E_n| = d_0$. Hence, the algorithm converges.

Thus, we conclude that the algorithm converges to a correct expression.□

4. Inferring a Subclass

In the previous section, the whole class of context-free languages was considered as the class of inference object. It is, however, obvious that the algorithm is not so efficient, and we know that guessing from several comlexity results on inferring regular sets, it seems to be very hard to find an efficient algorithm for the inference problem of this class of languages, neither.(Note that, for example, the minimum inference problem for finite-state automata is shown to be NP-complete.[Go78]) On the other hand, a recent report on the polynomial time algorithm for regular sets ([An86]) suggests a possibility of finding a practical inference algorithm, provided that a representation space and a target class of languages are carefully and successfully chosen.

In this section we restrict our attention to a subclass of context-free languages called semilinear languages, and examine the complexity results on the inference problem for the class as well as the algorithm for the problem. In course of examination, it turns out that a context-free expression approach to the inference problem for the subclass has a preferable property that an enumeration procedure in the inference algorithm can be achieved by a context-free grammar, which leads to the meta inference discussion in Section 5.

4.1 Semilinear Languages and Inference Algorithm

Definition 4.1(Semilinear languages [Gr71])

(i) A context-free grammar $G=(N,T,P,S)$ is *semilinear* if it satisfies the following: For each $A \in N$, $x \in (N \cup T)^*$, $A \Rightarrow^* x$ implies that x contains at most one occurrence of A.

(ii) A language L is *semilinear* if there exists a semilinear grammar G such that $L=L(G)$.

Remarks.

① There are a number of terminological equivalents used for semilinear grammars and languages : finite-index grammars and languages[Sa73], non-expansive grammars, derivation-bounded languages[GS68], superlinear grammars[Br68].

② Be careful not to confuse with another "semilinear" in the formal language theory, which usually indicates a set of vectors in Euclidean space.

Now, Let T be a finite terminal alphabet, and let $T_2=T \cup \{\sigma_1,\sigma_2\}$ and $\Gamma_2 = T \cup \{\sigma_1,\sigma_2, \boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2\} \cup \{\phi, +, (,)\}$.

Lemma 4.1 ([Gr71])

Any semilinear language over T *is contained in* Ω_{T_2}*, that is , for any semilinear language* L *over* T *there exists a context-free expression* E *over* T_2 *such that* $L=|E|$ *holds.*

This property on semilinear languages distinguishes the context-free expression method from any other representation method (such as grammars , or abstract machines). That is, in the grammatical representation, for example, there is no upper bound on the number of nonterminals neccessary for expressing arbitrary semilinear language. This is known as the fact that for any non-negative integer n, there exists a semilinear language L_n which cannot be generated by any context-free grammar whose number of nonterminals is less than n.

Lemma 4.2

Let EXP_2 *be the set of context-free expressions containing only symbols from* Γ_2*, that is,* $EXP_2=\{E \mid \phi \rightarrow^* E$ *and* E *consists of* $\Gamma_2\}$*. Then, there exists a context-free grammar* G *such that* $L(G)=EXP_2$ *holds.*

Proof.

Consider the following $G = (\{A\}, \Sigma, P, A)$, where $\Sigma = \Gamma_2 \cup \{(,), +\}$, P is defined by:

(1) $A \to AA$

(2) $A \to a$ ($\forall a \in T \cup \{\sigma_1, \sigma_2, \varepsilon, \phi\}$)

(3) $A \to (A)\sigma_i$ ($i = 1,2$)

(4) $A \to (A+A)$.

Let E be an arbitrary expression in EXP_2, then by induction on the length of an expression used in the proof of Lemma 3.1, it is shown that $A \Rightarrow^* E$ and $E \in \Sigma^*$. Conversely, suppose that $A \Rightarrow^n x$ and $x \in \Sigma^*$. Then, we shall show by the induction of n that x is in EXP_2. When $n = 1$, it is trivial. Suppose that the claim holds for $n \leqq k$ and that

$$A \Rightarrow^k w_1 A w_2 \Rightarrow w_1 w w_2 = x, \quad \text{where } w, w_1, w_2 \in \Sigma^*.$$

If this is reviewed as

$$A \Rightarrow y \Rightarrow^k x,$$

the first step $A \to y$ in the derivation is one of the rules (1), (3), (4) above. Let $y = AA$, then there exist i,j such that $A \Rightarrow^i x$, $A \Rightarrow^j x$, and $x = x_1 x_2$ with $i + j = k$. By the induction hypothesis, since $x_1, x_2 \in EXP_2$, we have that $x \in EXP_2$. The same argument works for the cases $y = (A)\sigma_i$ and $y = (A+A)$.□

Example 4.1 (Derivation Tree for an Expression)

Consider a language $L = (\{a^i b^i | i \geqq 1\} c)^*$. The language L is known as a semilinear language. An expression E satisfiying $L = |E|$ is derived by the grammar G as shown in Figure 3.

As a result, the derived expression E is :

$$E = ((a\sigma_2 b + ab)\sigma_2 c\sigma_1 + \varepsilon)\sigma_1.□$$

Thus, the enumeration procedure can be realized by a context-free grammar as a generator.

From these observations, define an operator δ_2 on EXP_2 as follows:

(1) $\phi \to \phi\phi \mid (\phi + \phi) \mid (\phi)\sigma_1 \mid (\phi)\sigma_2 \mid a$ ($a \in T \cup \{\sigma_1, \sigma_2, \varepsilon\}$)

(2) if $E_1 \to E$, then $E_1 + E_2 \to E + E_2$ and $E_2 + E_1 \to E_2 + E$

(3) if $E_1 \to E$, then $E_1\sigma_1 \to E\sigma_1$ and $E_1\sigma_2 \to E\sigma_2$

(4) if $E_1 \to E$, then $E_1 E_2 \to E E_2$ and $E_2 E_1 \to E_2 E$.

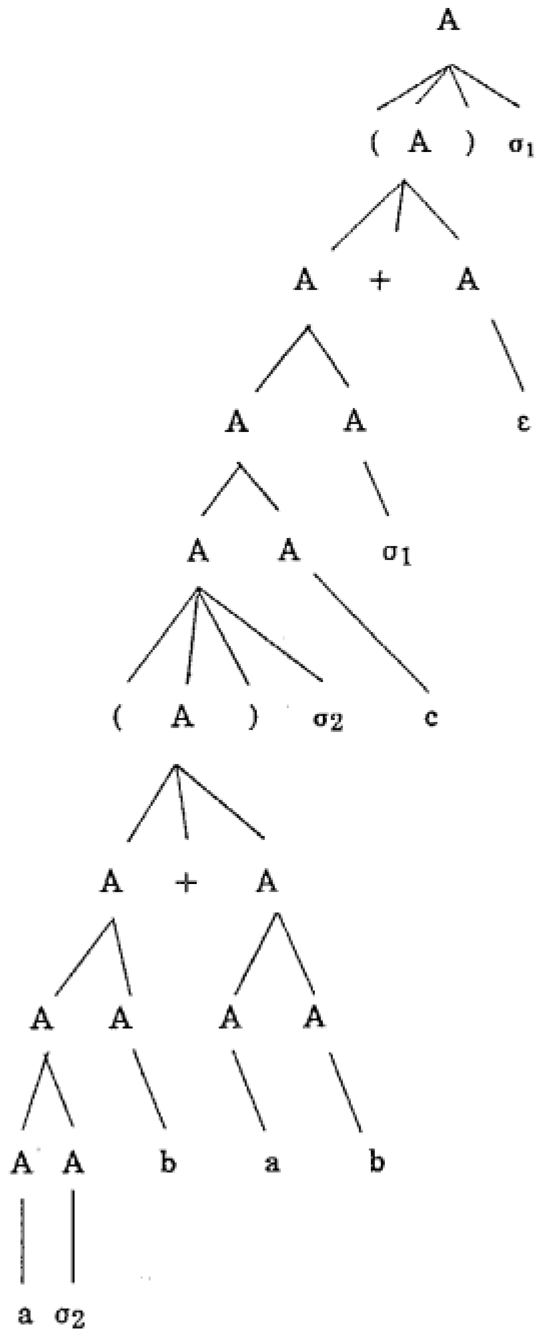

Figure 3. Derivation Tree for E

Notes. The rules (2),(3),(4) are required only for δ_2 to be a operator *on EXP_2*. Hence, the essential part of δ_2 consists of only (1), which is identified with a context-free grammar, as previously seen.

Lemma 4.3

The operator δ_2 defined above satisfies the following properties:

(i) *δ_2 is complete for the most specific expression ϕ in the sense that the set $\delta_2^*(\phi)$ of all expressions obtainable by a finite number of applications of δ_2 from ϕ includes at least one expression for every semilinear language.*

(ii) *For arbitrary expressions* E_1, E_2 *in* EXP_2, *whenever* $E_1 \in \delta_2(E_2)$, $|E_1| \supseteq |E_2|$ *holds.*

Proof. Obvious from the definitions and Lemmas 4.1 and 4.2.□

The inference algorithm for semilinear languages results in a very concise formulation, which is a special case of the one for context-free languages and a generalization of the case for regular expressions and is given as Algorithm A_2.

Theorem 4.1

For any semilinear language d_0, the algorithm A_2 identifies d_0 in the limit.

Proof. Similar to that of Theorem 3.3 and abbreviated.□

4.2 Another Algorithm

Although, in the previous subsection, a simple inference algorithm A_2 for semilinear languages has been given, it is too naive and ineffective to be used for practical purpose. Here we shall present another version of an inference algorithm.

Before going on to a formal discussion, let's see the follwing example which suggests the outline of the proof for the lemma coming up.

Example 4.2

Consider a linear grammar $G=(\{S,A,B\}, \{a,b,c\}, P, S)$, where $P = \{S \to Ab \mid bB \mid aS \mid b,\ A \to Sb \mid bB \mid aA,\ B \to bS \mid aA \mid Bc\}$. Suppose that we want to get an expression E such that $|E| = L(G)$. Let P_X be the set of rules with X on the left side. First, consider the following procedure: Let $A(\neq S)$ be a nonterminal such that the subset $R_A = \{A \to u_iB_iv_i \mid$ neither $B_i \neq S$ nor $\neq A (i=1,\dots,p)\}$ of P_A is non-empty. (If there is no

```
Algorithm A_2 (Inference Algorithm for Semilinear Languages)

Input: A recursively enumerable set of context-free expressions EXP_2
       An enumeration operator δ_2
       An admissible presentation of a target language d_0

Output: A sequence of expression E_1, E_2, ... such that E_n is correst for the first
        n examples.

Procedure:
    Q←Φ;{ possible expressions are stored in the queue Q}
    EXAMPLES←Φ (empty set)
    E←φ;{E keeps the current hypothesis; φ is the most specific expression}
    do (forever):
        EXAMPLES←EXAMPLES∪EX();{get next example}
        repeat
            if E is "too specific"
            then  append δ_2(E) to the tail of Q;
                  E←next(Q);{next removes the top element of Q}
            else  if E is "too general"
                  then  discard E ;
                        E←next(Q);
        until  E is correct for every example in EXAMPLES
        Output E as the next hypothesis.

where
   [E is "too specific"] and [E is "too general" ] are defined as in Algorithm A_1
```

such a nonterminal but S in N, then this procedure may be skipped, and move on to the next step below.) Further, let $X_1 \to x_1Ay_1, \ldots, X_n \to x_nAy_n$ be all productions whose right sides contain A. Then, by substituting all $u_iB_iv_i$ into A in the right side of each $X_j \to x_jAy_j$, construct $n \times p$ new productions $X_j \to x_ju_1B_1v_1y_j | \cdots | x_ju_pB_pv_py_j$ $(1 \leq j \leq n)$ and add them to P, and at the same time, remove R_A from P. It is clear that resulting new grammar G' is equivalent to the original G. In our case, $R_A = \{A \to bB\}$, $R_B = \{B \to aA\}$. If we take B and apply the above procedure to B, we have a modified grammar G' in which the set of productions P' is $\{S \to Ab \,|bB\, |aS\, |b,\ A \to Sb\, |bB\, |aA|\, baA,\ B \to bS\, |Bc\}$.

Construct $E_B=(bS+\sigma_2 c)\sigma_2$ for P'_B, and replace all occurrences of B appearing in $P' - P'_B$ with E_B, yielding a new grammar G_1 with the set of productions $\{S\rightarrow Ab\ |bE_B\ |aS\ |b,\ A\rightarrow Sb\ |bE_B|aA|baA\}$. It holds that $sub_B(L(G_1))=L(G')=L(G)$, where $sub_B(E_B)=|E_B|$. At this time, since R_A is empty for this new grammar, by constructing $E_A=(Sb+bE_B+a\sigma_2+ba\sigma_2)\sigma_2$, we have a modified grammar G" in which the set of productions is $\{S\rightarrow E_Ab|bE_B\ |aS|b\}$. Finally, we have an expression :

$$\begin{aligned}E_S&=(E_Ab+bE_B+aS+b)S\\ &=((Sb+b(bS+\sigma_2c)\sigma_2+a\sigma_2+ba\sigma_2)\sigma_2b+b(bS+\sigma_2c)\sigma_2+aS+b)S,\end{aligned}$$

By replacing S with σ_1, eventually

$$E_S=((\sigma_1b+b(b\sigma_1+\sigma_2c)\sigma_2+a\sigma_2+ba\sigma_2)\sigma_2b+b(b\sigma_1+\sigma_2c)\sigma_2+a\sigma_1+b)\sigma_1,$$

is obtained. From the way of construction, it is obvious that E_S is the desired one. □

Lemma 4.4

Let $G=(N,\Sigma,P,S)$ *be a reduced linear (context-free) grammar with the property that for any* $A,B\in N$, *there exist* $x,y,x'y'\in\Sigma^*$ *such that* $A\Rightarrow^* xBy$ *and* $B\Rightarrow^* x'Ay'$. *Then, there is an expression* E_S in EXP_2 *such that* (1) $L(G)=|E_S|$ *and* (2) E_S *is of the form* $(E(\sigma_1)+E(E_{A_1})+\cdots+E(E_{A_r})+E(\varepsilon))\sigma_1$, *where* $E_{A_i}=(E(\sigma_1)+E(\sigma_2)+E(E_{B_1})+\cdots+E(E_{B_t}))\sigma_2$ $(r,t\geqq 0,\ A_i,B_j\in N-\{S\})$, *and* $E(X)$ *is an expression schema of a finite summation of the form:* $\Sigma u_iXv_i\ (u_i,v_i\in\Sigma^*,\ i\geqq 0)$.

Proof. From the property of G, we may assume that $X\rightarrow\alpha\in P$ and $\alpha\in\Sigma^*$ imply $X=S$.(Otherwise, using the property one can modify G and get an equivalent grammar satisfying this condition.) Since the lemma is trivial in case of $N=\{S\}$, we may assume that N contains at least two nonterminals. For X in N, let P_X be the set of all productions with X on the left side. First, consider the following procedure:

[Procedure] Let $A(\neq S)$ be a nonterminal such that the subset $R_A=\{A\rightarrow u_iB_iv_i|$ neither $B_i\neq S$ nor $\neq A(i=1,\ldots,p)\ \}$ of P_A is non-empty. (If there is no such a nonterminal but S in N, then this procedure may be skipped, and move on to the next step below.) Further, let $X_1\rightarrow x_1Ay_1,\ldots,X_n\rightarrow x_nAy_n$ be all productions whose right sides contain A. Then, by substituting all $u_iB_iv_i$ into A in the right side of each $X_j\rightarrow x_jAy_j$, construct $n\times p$ new productions $X_j\rightarrow x_ju_1B_1v_1y_j|\cdots|x_ju_pB_pv_py_j\ (1\leqq j\leqq n)$ and add them to P, and at the same time, remove R_A from P. It is clear that resulting new grammar G' is equivalent to the original G. Note that since $R_A=\Phi$, P_A in the grammar G' must be of the form :

$\{A \to x_1Sy_1|\cdots|x_tSy_t|u_1Av_1|\cdots|u_rAv_r\}$, where $x_i,y_i,u_j,v_j \in \Sigma^* \cdots$ (#).

Now, construct an expression E_A from (#) as follows:

$E_A=(x_1Sy_1+\cdots+x_tSy_t+u_1\sigma_2v_1+\cdots+u_r\sigma_2v_r)\sigma_2$

and replace all occurrences of A appearing in $P-P_A$ with E_A, yielding a new set of productions:

$P'=\{X \to f(\alpha)|X \to \alpha \in P-P_A\}$, where f is a homomorphism defined by $f(Y)=Y(Y \neq A)$, $f(A)=E_A$. Consider a grammar $G_1=(N-\{A\}, \Sigma \cup \{E_A\}, P', S)$, then it is easy to see that $sub_A(L(G_1))=L(G')=L(G)$, where $sub_A(E_A)=|E_A|$, $sub_A(a)=a(a \in \Sigma)$.(End of Procedure)

After repeatedly applying the above procedure to all $A(\neq S)$ in N such that $R_A \neq \Phi$ until no such an A exists any more, let G_m be the final resulting grammar. (Note that the above procedure obviously terminates.). And, let $A_1,\ldots,A_m$ be a sequence of nonterminals involved in the elimination process from G to G_m above. Then, we have:

$sub_{A_1}(\ldots(sub_{A_m}(L(G_m))\ldots)=L(G)$, and in G_m,

for $A(\neq S)$ in N_m(the set of nonterminals of G_m) if any,

P_A is of the form:

$\{A \to x_1Sy_1|\cdots|x_tSy_t|u_1X_1v_1|\cdots|u_rX_rv_r\}$, where $x_i,y_i,u_j,v_j \in \Sigma^*$, $X_j \in \{E_B|B \in N-N_m-\{S\}\} \cup \{A\}$, and

P_S is of the form:

$\{S \to x_1Sy_1|\cdots|x_tSy_t|u_1A_1v_1|\cdots|u_rA_rv_r|z_1|\cdots|z_q\}$, where x_i, y_i, u_j, v_j, $z_k \in \Sigma^*$, $A_j \in N_m \cup \{E_A|A \in N-N_m-\{S\}\}$.

Then, for each A in $N_m-\{S\}$ by constructing an expression E_A in the manner above and substituting each E_A into A appearing in the right sides of P_S, we eventually have $E_S=(x_1Sy_1+\cdots+x_tSy_t+u_1E_{A_1}v_1+\cdots+u_rE_{A_r}v_r+z_1+\cdots+z_q)S$. Finally, by replacing S with σ_1,

$E_S=(x_1\sigma_1y_1+\cdots+x_t\sigma_1y_t+u_1E_{A_1}v_1+\cdots+u_rE_{A_r}v_r+z_1+\cdots+z_q)\sigma_1$

is obtained. Thus, using a schema $E(X)=\Sigma x_iXy_i$, we have a desired form of an expression E_S. It is almost obvious that $|E_S|=L(G)$ holds.□

Lemma 4.5

Let $G=(N,\Sigma,P,S)$ *be a reduced semilinear (context-free) grammar. Then, there is an expression* E_S *in* EXP_2 *such that* (1) $L(G)=|E_S|$ *and* (2) E_S *is of the form* $(E(\sigma_1)+E(E_{A_1})+\cdots+E(E_{A_r})+E(\varepsilon))\sigma_1$, *where* $E_{A_i}=(E(\sigma_1)+E(\sigma_2)+E(E_{B_1})+\cdots+E(E_{B_t})+E(\varepsilon))\sigma_2$ $(r,t\geqq 0$, and $A_i,B_j\in N-\{S\})$, *and* $E(X)$ *is an expression schema of a finite summation of the form*: $\Sigma u_iXv_i\ (u_i,v_i\in(\Sigma\cup\{E_A|A\in N-\{S\}\})^*$, *and* $i\geqq 0)$.

Proof. It is known ([Gr71])that a context-free grammar $G=(N,\Sigma,P,S)$ is semilinear if and only if all grammatical levels of G is linear, where a *grammatical level* G_i of G is an ordered pair (N_i, P_i), each P_i is an equivalence class of $P/\equiv$: $A\rightarrow\alpha\equiv B\rightarrow\beta$ iff either $A=B$ or $[A\Rightarrow^*x_1By_1$ and $B\Rightarrow^*x_2Ay_2$, for some $x_1,x_2,y_1,y_2\in(N\cup\Sigma)^*]$, and N_i is the set of symbols on the left sides of productions in P_i. A grammatical level is *linear* iff the set of productions is linear. Let $G_0=(N_0,P_0)$ be the grammatical level such that N_0 contains S, and $G_k=(N_k,P_k)(1\leqq k\leqq t)$ be all other grammatical levels of G. Then, define a binary relation $\vdash$ on $P/\equiv$ as follows: $G_i\vdash G_j$ iff there is $A\rightarrow xBy$ in P_i, for some A in N_i, B in N_j, $x,y\in(N\cup\Sigma)^*$. In this case B is regarded as a terminal symbol(called *quasi-terminal*) in G_i and denoted by its boldface $\mathbf{B}$. A relation $\vdash$ introduces into $P/\equiv$ a semi-lattice structure with the maximum element G_0 .(Note that G is assumed to be reduced.) Further, all grammatical levels $G_j=(N_j, P_j)$ such that $A\rightarrow\alpha\in P_j$ implies $\alpha\in(N_j\cup\Sigma)^*$ constitute minimal elements. More precisely, since we are dealing with a semilinear grammar G (i.e., G_j is linear), α must be in $\Sigma^*N_j\Sigma^*\cup\Sigma^*$. Hence, by Lemma 4.4 for each X in N_j we can construct an expression $E_{G_{j,X}}$ which is in EXP_2 and $|E_{G_{j,X}}|=L(G_{j,X})$, where $G_{j,X}=(N_j,\Sigma,P_j,X)$.

Now, suppose that $G_j\vdash G_{ji}(i=1,\ldots,p)$ hold and that for each X in N_{ji} and G_{ji}, $E_{G_{ji,X}}$ has been obtained. Then, since each symbol X in N_{ji} is a quasi-terminal $\mathbf{X}$ in $G_j=(N_j,P_j)$ and G_j is linear, by replacing $\mathbf{X}$ with $E_{G_{ji,X}}$ in G_j, for each A in N_j we can get an expression $E_{G_j,A}$, and $|E_{G_{j,A}}| = f(L(G_{j,A}))$ holds, where $G_{j,A} = (N_j, \Sigma \cup \{X | X \in N_{ji} (i=1,...,p)\}, P_j, A)$ and $f(X) = |E_{G_{ji,X}}| (X \in N_{ji})$, $f(a) = a\, (a \in \Sigma)$. By applying this procedure to each node of the semi-lattice recursively in a bottom-up manner, we can eventually have an expression $E_{G_0,S}$ for S in N_0 and G_0. (Note that G_0 is the unique element on the top of the semi-lattice). From the way of construction, it is easy to prove that $|E_{G_0,S}| = L(G)$ holds. (For example, this can be proved by the induction on the height, i.e, the length of the longest path from the top to the bottoms, of the semi-lattice.) It is also easily seen that the expression $E_{G_0,S}$ is in the desired form. □

Example 4.3

Returning to Example 4.1, consider a semilinear grammar $G=(\{S,A\}, \{a,b,c\}, \{S \to AcS|\varepsilon,\ A \to aAb|ab\}, S)$ generating a language $L=(\{a^ib^i | i \geq 1\}c)^*$. Then, grammatical levels of G are : $G_0=(\{S\}, \{S \to AcS|\varepsilon\})$, $G_1=(\{A\}, \{A \to aAb|ab\})$, and the semi-lattice structure of G is $G_0 \vdash G_1$. For A and G_1, we have an expression: $E_{G_1,A} = (a\sigma_2 b + ab)\sigma_2$, and hence for S and G_0, $E_{G_0,S} = ((a\sigma_2 b + ab)\sigma_2 c\sigma_1 + \varepsilon)\sigma_1$ is obtained.□

Now, from these observations, we can refine an operator δ_2 and modify an inference algorithm A_2. First, we extend the alphabet over which expressions are defined. Let $\Delta = \{X_S, Y_S, X_1, X_2, Z_1, Z_2, W_1, W_2\}$ and Δ' be its boldface version. Define an operator δ_2' on the set of expressions over $\Delta \cup T_2$ as follows:

(1) $\phi \to (X_S + X_1)\sigma_1$

(2) $X_1 \to Y_S + X_1 | W_1$

(3) $W_1 \to W_1 + W_2 | \varepsilon$

(4) $W_2 \to aW_2 | a \ (a \in T \cup \{\varepsilon\})$

(5) $X_S \to X_2\sigma_1 X_2 + X_S | \varepsilon$

(6) $X_2 \to Y_S X_2 | aX_2 | Y_S | a \ (a \in T \cup \{\varepsilon\})$

(7) $Y_S \to (X_S + Z_1 + Z_2)\sigma_2$

(8) $Z_1 \to X_2\sigma_2 X_2 + Z_1 | \varepsilon$

(9)$Z_2 \rightarrow Y_S + Z_2 | X_2$

(10) if $E_1 \rightarrow E$, then $E_1 + E_2 \rightarrow E + E_2$ and $E_2 + E_1 \rightarrow E_2 + E$

(11) if $E_1 \rightarrow E$, then $E_1{}^{\sigma_1} \rightarrow E^{\sigma_1}$ and $E_1{}^{\sigma_2} \rightarrow E^{\sigma_2}$

(12) if $E_1 \rightarrow E$, then $E_1E_2 \rightarrow EE_2$ and $E_2E_1 \rightarrow E_2E$.

Note that no boldface symbol of Δ actually appears in any expression involved in the extension here.

Then, we can prove the following lemma.

Lemma 4.6

The operator δ_2' defined above satisfies the following properties: δ_2' is complete for the most specific expression ϕ in the sense that the set $\delta_2'{}^*(\phi)$ of all expressions obtainable by a finite number of applications of δ_2' from ϕ includes at least one expression for every semilinear language.

Proof. From Lemma 4.5 , let $E_S = (E(\sigma_1) + E(E_{A_1}) + \cdots + E(E_{A_r}) + E(\varepsilon))^{\sigma_1}$ be an arbitrary expression for a semilinear language, where each component satisfies the condition stated in Lemma 4.5. It suffices to see that there is a derivation such that $\phi \rightarrow^* E_S$. For this end, all one has to do is to see that $X_S \rightarrow^* E(\sigma_1)$ and $X_1 \rightarrow^* E(E_{A_1}) + \cdots + E(E_{A_r}) + E(\varepsilon)$, and hence, that $Y_S \rightarrow^* E(E_{A_i})$ and $W_1 \rightarrow^* E(\varepsilon)$. (Formal discussion is omitted here.) □

Now, a modified version A_2' of the algorithm A_2 is straightforwardly obtained by replacing δ_2 with δ_2' and making a small modification to restrict the domain of expressions to EXP_2.

Theorem 4.2

For any semilinear language d_0, the algorithm A_2' identifies d_0 in the limit.

Proof. Almost obvious from Lemma 4.6.□

4.3 Complexity Results of Inference Problems

In this subsection the following type of problem is considered: For a given sample set S, find a context-free expression E such that E is compatible with S, Further, another type of inference problem of deciding whether or not for a given

```
Algorithm A_2' (Improved Algorithm for Semilinear Languages)

Input: A recursively enumerable set of context-free expressions $EXP_2$
       An enumeration operator $\delta_2'$
       An admissible presentation of a target language $d_0$

Output: A sequence of expression $E_1, E_2, \ldots$ such that $E_n$ is correst for the first
        n examples.

Procedure:
    $Q \leftarrow \delta_2'(\phi)$;{ possible expressions are stored in the queue Q,
                $\phi$ is the most specific expression}
    EXAMPLES$\leftarrow\Phi$ (empty set)
    $E \leftarrow$get(Q);{E keeps the current hypothesis,
                next removes the top element of Q}
    do (forever):
        EXAMPLES$\leftarrow$EXAMPLES$\cup$EX();{get next example}
        repeat
            if E is not in $EXP_2$
            then append $\delta_2'(E)$ to the tail of Q;
                 $E \leftarrow$next(Q);
            else (E is in $EXP_2$)
                if E is "too specific" or E is "too general"
                then discard E ;
                     $E \leftarrow$next(Q);
        until E is correct for every example in EXAMPLES
        Output E as the next hypothesis.

where
  [E is "too specific"] and [E is "too general" ] are defined as in Algorithm $A_1$
```

sample set S and a positive integer t there exists an expression E whose size is t and is compatible with S is also examined.

Definition 4.2

(1) The *size* of a context-free expression is the number of occurrences of the symbols of the alphabet.

(2)A *sample set* S over Σ is a finite subset of $\Sigma^* \times \{+,-\}$ such that whenever (u,a) and (v,b) are members of S and u=v, then a=b.

(3) A context-free expression E *is compatible with* a sample set S iff for each (u,a) in S, $u \in |E|$ iff $a = +$.

Definition 4.3

Let S be a sample set over Σ, and t be a positive integer. Further, let Ξ be a subcalss of the class of context-free languages and Ω be the smallest set of context-free expressions sufficient to express any language in Ξ. Moreover, it is assumed that an enumeration procedure *P* for elements of Ω is given. Then,

(1) the *minimum inference problem* for Ξ via *P* is to find the minimum index of an expression E in the enumeration *P* that is compatible with S, and

(2) the *inference decision problem* for Ξ is to decide whether or not there exists an expression E whose size is t and is compatible with S.

Theorem 4.3

Let P be an enumeration procedure with the order in size. Then, the minimum inference problem for the class of semilinear languages is NP-*hard.*

Proof. First, it is known that the minimum inference problem for regular sets in terms of regular expressions is NP-hard([An78]). Further, we can show that for any regular expression E, one can construct a context-free expression E' such that E' is in EXP_2 and $|E| = |E'|$. In fact, if E is of the form $(E_1)^*$(or $(E_1)^+$), then let E'be $(E_1\sigma+\varepsilon)\sigma$(or $(E_1\sigma+E_1)\sigma$). If E is of the form E_1+E_2 (or E_1E_2), then let E' be E itself. Apply this procedure to E_i recursively. It is almost obvious that the resulting expression E' is in EXP_2 (actually in EXP_1:the set of expression using only one auxiliary symbol) and $|E| = |E'|$. This takes at most 0(n) time, where n is the size of an expression E. Thus, the minimum inference problem for regular sets in terms of regular expressions is polynomially reducible to this problem at issue, hence the problem is NP-hard.□

Thus, it is seen that the minimum inference problem in which the enumeration procedure in the algorithm A_2 is employed is NP-hard.

Theorem 4.4

The inference decision problem for the class of semilinear languages is NP-*complete.*

Proof. It sufices to show the following two: Let R={(S,t)| there is an expression E in EXP_2 whose size is at most t and is compatible with S}, then (i) R is in NP(the class of nondeterministic languages polynomially solvable), and (ii) R is NP-hard.

The proof of (i): By Lemma 4.2, each element in EXP_2 is generated by a specific context-free grammar G. Since, except for the rules in (2), the right-hand side of each rule in G contains at least two symbols, the height of a derivation tree for an expression E in G is at most 0(t). Hence, we can nondeterministically guess the expression E of length t, which takes at most 0(t). Further, let m be the cardinality of S and n=max{lg(u) | (u,a) ϵS}, then it takes at most $0(mn^3)$ to see if for each (u,a) in S, u϶|E| iff a= +. (Note that the membership problem for context-free grammars is solvable in time $0(n^3)$.[HU69]) Hence, R is in NP.

The proof of (ii):By [An78], it is known that the inference decision problem for regular sets in terms of regular expressions is NP-hard. Further, as shown in the proof of the previous theorem, from a given regular expression E it is possible to construct an equivalent expression E' in EXP_2 in polynomial time. Hence, R is NP-hard. This completes the proof.□

5. Meta Inference

In the previous section, we have seen that a context-free grammar can work as a generator for enumerating expressions in the inference algorithm for semilinear languages. This observation leads to the following interesting topic on meta inference algorithms.

Reviewing the inductive inference problem for formal languages, the inference model states : Given an object L (context-free language), an inductive inference device (IID) tries to infer a representation(context-free expression) denoting the object L from its examples. This is illustrated by Figure 4 below.

On the other hand, as we have seen, an expression enumerator δ is realized as a context-free grammar, which implies that an enumerator in the IID is identified with

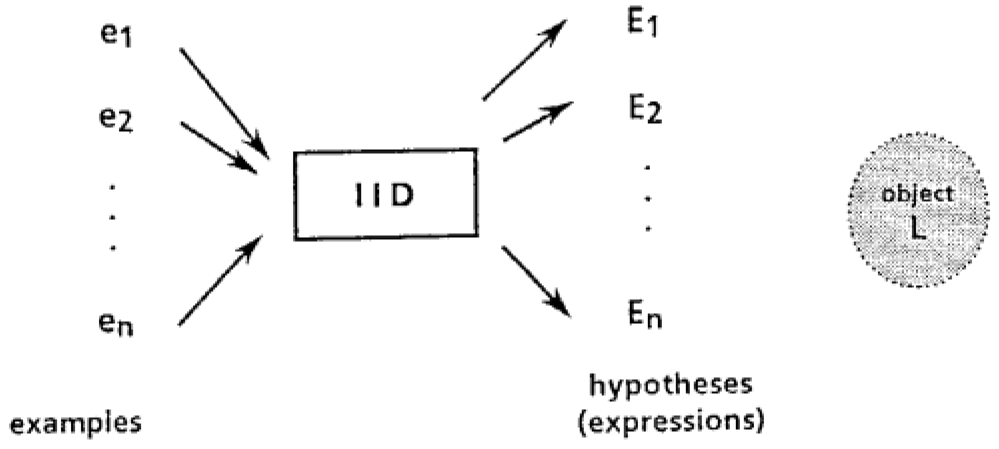

Figure 4. Inductive Inference Schema

a context-free expression. Therefore, one can think of a meta inference problem in which for a given object, the meta-IID infers a representation (context-free expression, or enumerator) denoting the object from examples of expressions, where the object is a class of expressions (or, a class of context-free languages denoted by the expressions). Note that since the input of meta-IID, which is an expression, can be regarded as a string over some alphabet, the inference schema of the meta-inference problem is structurally equivalent to that of the inference problem in the usual sense, which is illustrated by Figure 5.

Hence, we can discuss the meta inference problem for a family (or hierarchy) of language classes within the semilinear context-free languages and easily get an inference algorithm A_3 as shown below. In the problem setup, the object is given as a target class C of languages (e.g., the class of semilinear languages, the class of linear context-free languages, or the class of regualr languages, etc.). The sequence of examples consists of context-free expressions denoting the corresponding context-free languages on C. The inference algorithm produces a sequence of hypothesiss (enumerators), by each of which any language in C could be possibly infered. The enumerator δ_2 can be employed as a meta enumerator for the algorithm A_3.

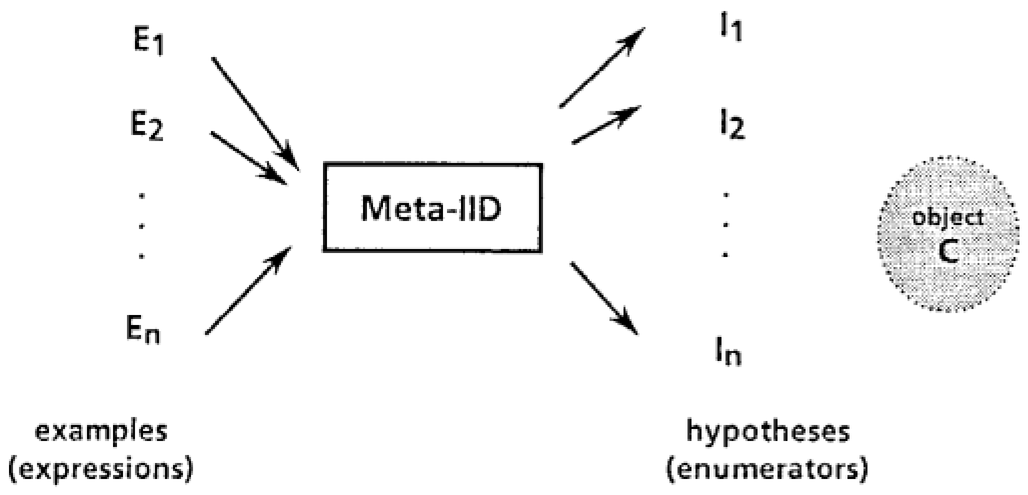

Figure 5. Meta Inductive Inference Schema

6. Concluding Remarks

Using context-free expressions, we have considered the inductive inference problem for context-free languages. The context-free expression method we developed here may be characterized by the fact that compared with the conventional problem setup of grammatical inference, it can provide a very simple inference algorithm. The method has a great advantage that the regular expression approach may be naturally extended to the case.

There are, generally, two types of inference procedures : enumerative methods and constructive methods. Although this classification is not clear for some class of inference methods already proposed, enumerative methods have an advantage over constructive mehtods in that since the former is based on the exhaustive search strategy, it is complete and optimal in some sense. While the latter is typically more effective than the former.

Now, there are a number of literatures which deal with the inductive inference problem for context-free languages. In [KK76], a constructive method for grammatical inference of context-free language(more exactly, BNF expressions) is

```
Algorithm $A_3$ (Meta Inference Algorithm for a family of language classes)
Input: A recursively enumerable set of context-free expressions $EXP_2$
       An enumeration operator $\delta_2$
       An admissible presentation of the target class C of languages

Output: A sequence of enumerators (expressions)$I_1$, $I_2$, ... such that $I_n$ is
       correst for the first n expressions on C.

Procedure:
       Q←Φ;{ possible enumerators are stored in the queue Q}
       EXAMPLES←Φ (empty set)
       I←ϕ;{I keeps the current hypothesis; ϕ is the most specific
             enumerator}
       do (forever):
          EXAMPLES←EXAMPLES∪EX();{get next expression}
          repeat
             if I is "too specific"
             then  append $\delta_2$(I) to the tail of Q;
                   I←next(Q);{next removes the top element of Q}
             else  if I is "too general"
                   then  discard I;
                         I←next(Q);
          until  I is correct for every expression in EXAMPLES
          Output I as the next hypothesis.

where
   [I is "too specific"] and [I is "too general" ] are defined as in Algorithm $A_1$
```

discussed from the programming language inference point of view. [Wh] discusses enumeration procedures of context-free languages, and presents several techniques for improving efficiency of the procedure. Due to being based on the constructive method, the former seems not to have a theoretical background such as the completeness, while the latter employs many improvements, but is still far from the practical efficiency. [Ta86] proposes a constructive method for composing grammars based on the idea of extending linear grammar case. Although, by assuming the upper bounds for making search space finite, this constructive method assures the completeness, none of the criteria for the precondition is discussed.

The context-free expression approach proposed in this paper, which is based on the enumerative method, has the advantage mentioned above. However, it still remains left open to solve the efficiency problem. It seems that we need to employ a kind of statistic ideas to get a more practical algorithm for language classes larger than regular sets.([An86],[Ki86],[Va84])

Acknowledgements

The author is grateful to Dr. T. Kitagawa, the president of IIAS-SIS, for ceaseless encouragements. He is also indebted to Dr. H.Enomoto, the director of IIAS-SIS, for providing useful reference papers as well as invaluable advice. Dr. T.Nishida gave many useful comments on the earlier draft of this paper. Last but not least, discussion with the colleagues Y.Takada, Y.Sakakibara, and H. Ishizaka was very fruitful.

This work is a part of the major research and development of FGCS Project conducted under the program setup by MITI.

References

[An78] Angluin,D., On the Complexity of Minimum Inference of Regular Sets, *inf. and Contr.* 39, 337-350 (1978).

[An86] Angluin,D., "Learning regular sets from queries and counter-examples", Technical Report 464, Dept. of Comput. Sci., Yale University, 1986.

[Bi72] Biermann,A.W., An Interactive Finite-State Languages Learner, *Proc. of the First USA-Japan Comput. Conf.*,13-20 (1972).

[Br68] Brzozowski,Y.A., Regular-like Expressions for Some Irregular Languages, *IEEE Conf. record of 9th Ann. Sym. on Switching and Automata Theory* 278-280, 1968.

[ET76] Enomoto,H. and Tomita,E., A Representative Set of Deterministic Finite Automata, *Transactions of IECE*, Vol.59-D, No.9, 660-667 (1976).(in Japanese)

[Go67] Gold,E.M., Language Identification in the Limit, *Inf. and Contr.* 10, 447-474 (1967).

[Go78] Gold,E.M., Complexity of Automaton Identification from given Data, *Inf. and Contr.* 37, 302-320 (1978).

[Gr71] Gruska,J., A Characterization of Context-free Languages, *J. of Comput. and Sys. Sci.* 5, 353-364 (1971).

[GS68] Ginsburg,S. and Spanier,E.H., Derivation-Bounded Languages, *J. of Comput. and Sys. Sci.* 2, 228-250 (1968).

[HU69] Hopcroft,J.E. and Ullman,J.D., "Formal Languages and Their Relation to Automata", Addison-Wesley, 1969.

[Is86] Ishizaka,H., Model Inference Incorporating Generalization, *Proc. of Symp. on Software Sci. and Engineering*, Kyoto, Sept. 1986.

[Ki86] Kitagawa,T., "Statistical Information in Inference Process and Data Analysis", Research Report 66, Intern. Inst. for Advanced Study of Soc. Inform. Sci., 1986.

[KK77] Knobe,B. and Knobe,K., A method for Inferring Context-free Grammars, *Inf. and Contr.* 31, 129-149 (1976).

[La86] Laird,P.D., "Inductive Inference by Refinement", Technical Report TR-376, Dept. of Compt. Sci., Yale University, 1986.

[Sa73] Salomaa,A., "Formal Languages", Academic Press, 1973.

[Sh81] Shapiro,E., "Inductive Inference of Theories from Facts", Technical Report 192, Dept. of Comput. Sci., Yale University, 1981.

[Sh82] Shapiro,E., "Algorithmic Program Debugging", Ph.D Dissertation, Dept. of Comput. Sci., Yale University, 1982, also published by MIT Press, 1983.

[Shi83] Shinohara,T., Polynomial Time Inference of Extended Regular Pattern Languages, *Lecture Notes in Comput. Sci.* 147, Springer-Verlag, 115-127 (1983).

[Ta86] Tanatsugu,K. , A Grammatical Inference based on Self-embedding for Context-free Languages, *Proc. of LA Symp.*, Kyoto, Feb., 1986.(in Japanese)

[TA77] Tanatsugu,K. and Arikawa,S., On Characteristic Sets and Degrees of Finite Automata, *Intern. J. of Comput. and Inf. Sci.* 6, No.1, 83-93 (1977).

[Va84] Valiant,L.G., A Theory of the Learnable, *C.A.C.M.*, VOl.27, No.11, 1134-1142(1984).

[Wh77] Wharton,R.M., Grammar Enumeration and Inference, *Inf. and Contr.*33, 253-272 (1977).