Mathematical Analysis of Lexical-Functional Grammars

—Complexity, Parsability, and Learnability—

Tetsuro Nishino

In order to specify the syntax of human languages, J. Bresnan and R. M. Kaplan introduced Lexical-Functional Grammars (LFGs) extending context-free grammars (CFGs) by attaching functional equations to each production of CFGs. The LFG theory has been advanced with the aim of meeting the dual demands of parsability and learnability. R. C. Berwick has shown, however, that the membership problem for LFGs is NP-hard and that the LFG theory must be tightened if one wants to avoid computational intractability. In this paper, we first show another formal properties of LFG theory suggesting its computational intractability, which are stated as follows : (1) the emptiness problem for LFGs is undecidable, and (2) the membership problem for LFGs which have one ε-production is *EXPTIME*-hard. Based on this observation, we introduce Frontier-to-Root LFGs (FRLFGs) by restricting the form of functional equations attached to phrase structure rules. Then we show that the membership and parsing problems for FRLFGs can be solved in $O(n^2)$ time if the underlying CFG is unambiguous. Since the famous non context-free language $\{a^k b^k c^k \mid k \geq 1\}$ can be generated by an unambiguous FRLFG, we obtain that the membership and parsing problems for this language can be solved in $O(n^2)$ time. We also show that, given a string x of the length n, the membership and parsing problems for an FRLFG G can be solved in $O(n^3 + d(x, G) \cdot n)$ time, where $d(x, G)$ is the degree of ambiguity of x with respect to G. Furthermore, we show that there is a polynomial time algorithm to learn the set of annotated phrase structure rules of an unknown FRLFG using structural membership, structural equivalence, and structural output queries. Thus, we propose a sufficiently large subclass of LFGs meeting the dual demands of parsability and learnability from computational theoretic points of view.

1. Introduction

In order to specify the syntax of human languages, R. M. Kaplan and J. Bresnan introduced Lexical-Functional Grammars (LFGs) extending context-free grammars (CFGs) by attaching functional equations to each production of CFGs (Kaplan & Bresnan (1982)). Since then, LFGs have been widely used in linguistics, cognitive science, and natural language processing (Bresnan (ed.) (1982), Pinker (1982, 1984), Savitch et al. (eds.) (1987), Sells (1985), Winograd (1983)). The LFG theory has been advanced with the aim of meeting the dual demands of parsability and learnability. In Berwick (1982), however, R. C. Berwick showed that the membership problem for LFGs is NP-hard and mentioned that the LFG theory must be tightened if one wants to avoid computational intractability.

In this paper, we first show the following other results which suggest the computational intractabilities of LFGs : (1) the emptiness problem for LFGs is undecidable, and (2) the membership problem for LFGs which have one ε-production is *EXPTIME*-hard.

Based on the above observations, we propose a new subclass of LFGs meeting the dual demands of parsability and learnability from computational theoretic points of view. That is, we introduce *Frontier-to-Root LFGs* (FRLFGs for short) by restricting the form of functional equations attached to phrase structure rules. It is shown that the class of languages generated by FRLFGs properly includes the class of context-free languages and is included in the class of context-sensitive languages.

Concerning the parsing of FRLFG languages, we show that the membership and parsing problems for FRLFGs can be solved in $O(n^2)$ time if the underlying CFG is unambiguous. Since it is also shown that a famous non context-free language $\{a^k b^k c^k \mid k \geq 1\}$ can be generated by an unambiguous FRLFG, we obtain that the membership and parsing problems for this language can be solved in $O(n^2)$ time by using the FRLFG framework.

While, concerned with the learning of FRLFGs, we first show that a translation realized by the set of annotated phrase structure rules of an FRLFG can be defined in terms of a tree transduction realized by a deterministic frontier-to-root skeletal automaton with output. Then, using this characterization, we show that there is a polynomial time algorithm to learn the set of annotated phrase structure rules of an unknown FRLFG using structural membership, structural equivalence, and structural output queries.

Our learning protocol is based on Angluin's *minimally adequate teacher* (Angluin (1987)). Further, our algorithm is an extension of Sakakibara's learning algorithm for deterministic tree acceptors in (Sakakibara (1988)) to the one for deterministic tree transducers.

Thus, we propose a subclass of LFGs meeting the dual demands of parsability and learnability from computational theoretic points of view.

2. Lexical-Functional Grammars

In this section, we review the definition of lexical-functional grammars. We only review the devices of LFG which are needed in the rest of this paper. LFG contains many other devices that facilitate linguistic analysis. For details, see Kaplan & Bresnan (1982).

In a sentence description of LFG, there are two types of components, a *constituent structure* (*c-structure*) and a *functional structure* (*f-structure*). Using these two structures, an LFG system specifies a set of grammatical sentences. A c-structure is a standard parsing tree of a context-free grammar. While an f-structure is a hierarchical structure constructed by the pair of function names and their unique values. The f-structure mainly represents the configuration of the surface grammatical functions.

An LFG is specified by a set of *annotated phrase structure rules*. An annotated phrase structure rule is a context-free rule associated with functional equations.

Definition 1 A *lexical-functional grammar* (LFG for short) is a 6-tuple $G = (NA, TA, S, FN, FV, AR)$ consists of 1-6 as follows :

1. NA is a finite *nonterminal alphabet.*
2. TA is a finite *terminal alphabet.*
3. S is a *start symbol.*
4. FN is a finite set of *function names.*
5. FV is a finite set of *function values.*
6. AR is a finite set of *annotated phrase structure rules.* An annotated phrase structure rule is of the form

$$A \rightarrow (B_1, E_1)\ (B_2, E_2) \cdots (B_m, E_m),$$

where $A, B_1, B_2, \cdots, B_m (m \geq 0)$ are nonterminal or terminal symbols and $E_i (1 \leq i \leq m)$ are sets of functional equations. We assume that at least one B_i is a nonterminal symbol. A *functional equation* is a statement of the form

$$M_1 F_1 F_2 = M_2 F_3 F_4,$$

where $M_1, M_2 \in \{\uparrow, \downarrow\}$ are called *metavariables*, and F_i $(1 \leq i \leq 4)$ are function names or function values. Any symbol except the sign of equality (i. e. "$=$") may be null, but there must be at least one symbol on each side of the equation. An annotated phrase structure rule of the form

$$A \rightarrow (b, E),$$

is especially called a *lexical insertion rule*, where $b \in TA$ and E is a set of functional equations which involve no $\downarrow$'s.

For an LFG $G = (NA, TA, S, FN, FV, AR)$, the following CFG Gr is called the *underlying context-free grammar* of G : $Gr = (NA, TA, S, P)$, where

$$P = \{A \rightarrow B_1 B_2 \cdots B_m \mid A \rightarrow (B_1, E_1)\ (B_2, E_2) \cdots (B_m, E_m) \in AR\}.$$

We assume that, for any LFG G, the underlying CFG Gr is λ-free and has no $X_i \rightarrow X_j$ type productions.

Example 1 (Kaplan & Bresnan (1982)) Let us consider the following simple LFG.

$G_1 = (NA, TA, S, FN, FV, AR)$, where

$NA = \{S, A, B, C\}$, $TA = \{a, b, c\}$, $FN = \{K\}$, $FV = \{0\}$,

AR involves the following rules :

$S \rightarrow (A, \{\uparrow = \downarrow\})\ (B, \{\uparrow = \downarrow\})\ (C, \{\uparrow = \downarrow\})$,

$A \rightarrow (a, \emptyset)\ (A, \{(\uparrow K) = \downarrow\})$,

$B \rightarrow (b, \emptyset)\ (B, \{(\uparrow K) = \downarrow\})$,

$C \rightarrow (c, \emptyset)\ (C, \{(\uparrow K) = \downarrow\})$,

$A \rightarrow (a, \{(\uparrow K) = 0\})$,

$B \rightarrow (b, \{(\uparrow K) = 0\})$, and

$C \rightarrow (c, \{(\uparrow K) = 0\})$.

The underlying CFG Gr_1 of G_1 is as follows:

$Gr_1 = (NA, TA, S, P)$, where

$P = \{S \rightarrow ABC,\ A \rightarrow aA,\ B \rightarrow bB,\ C \rightarrow cC,\ A \rightarrow a,\ B \rightarrow b,\ C \rightarrow c\}$.

A c-structure is generated using annotated phrase structure rules ignoring the functional equations appearing in them. Fig. 1 illustrates an example of a c-structue generated by the LFG G_1 (The meaning of the variables x_i, $1 \leq i \leq 12$ in Fig. 1 will be explained later.). An *annotated phrase structure tree* is a structure produced by attaching functional equations to a c-structure.

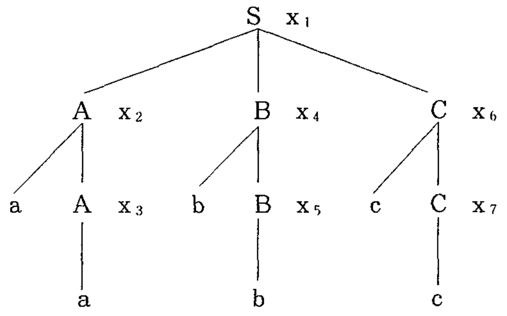

Figure 1 : An example of a c-structure generated by the LFG G_1.

Each internal node in a c-structure is assumed to have an associated f-structure. A metavariable $\downarrow$ attached to the node n in a c-structure represents the f-structure associated to n. While a metavariable $\uparrow$ attached to n represents the f-structure associated to the parent of n. Therefore a functional equation of the form $\uparrow = \downarrow$ attached to n represents that the f-structure associated to n is equal to the f-structure associated to the parent of n. While a functional equation of the form $(\uparrow\ K) = \downarrow$ attached to n represents that the f-structure associated to n is equal to the value for the function name K of the f-structure associated to the parent of n.

An f-structure is a hierarchical structure constructed by the pair of function names and their unique values as shown in Fig. 3 (a) or (b). Given an annotated phrase structure tree t, the f-structure corresponding to t is determined by the following procedure:

1. Assign a unique variable to each internal node of t which represents the f-structure associated to the node.
2. Using the variables of the step 1, *instantiate* (i. e. have the appropriate variables filled in for the arrows) all the equations associated to the nodes in t. The set of equations thus produced is called the *f-description* of t. (The f-description produced from the c-structure in Fig. 1 is shown in Fig. 2.)
3. Solve the f-description of the step 2 algebraically to obtain the value in the f-description's minimal solution of the $\downarrow$-variable associated to the S node. (This value is called the f-structure *assigned to the string.*) The outline of this step is described in the next paragraph.

$$\begin{array}{lll} x_1 = x_2, & (x_2\,K) = x_3, & (x_3\,K) = 0, \\ x_1 = x_4, & (x_4\,K) = x_5, & (x_5\,K) = 0, \\ x_1 = x_6, & (x_6\,K) = x_7, & (x_7\,K) = 0 \end{array}$$

Figure 2. The f-description produced from the c-structure in Fig. 1.

We now describe the outline of an f-description solution algorithm used in the above step 3. Since this algorithm was introduced in Kaplan & Bresnan (1982), we call this algorithm the *Kaplan-Bresnan* (*KB* for short) *algorithm*. Let us consider the following three functional equations in Fig. 2.

$$\begin{array}{rcl} (x_3\,K) & = & 0 \\ (x_2\,K) & = & x_3 \\ x_1 & = & x_2 \end{array}$$

The first equation means that x_3 is a partial function defined on the set of function names, which is at least defined on K, i. e. $x_3(K) = 0$. From this equation, we obtain the f-structure shown in Fig. 3 (a) as a tentative value for x_3. Then if we process the second equation, the tentative value for x_2 will be set to the one shown in Fig. 3 (b). Then if we process the third equation, a tentative value for x_1 will also be set to the f-structure shown in Fig. 3 (b). In the same way, we finally obtain the f-structure shown in Fig. 3 (b) as the value for x_1 of the f-description in Fig. 2. This f-structure is the one assigned to the string *aabbcc*. As mentioned in Kaplan & Bresnan (1982), the final result does not depend on the order in which equations are processed.

$[K\ 0]$ $\qquad$ $[K\ [K\ \ 0]]$

(a) $\qquad$ (b)

Figure 3. Simple f-structures.

Notice that if in an f-structure f is a solution for a given f-description, then any f-structure formed from f by adding values for function names not already present will also be a solution for the f-description. We say that in an f-structure f is a *minimal solution* for an f-description if f is a solution for it and if no smaller f-structure is a solution for it. A minimal solution exists for every solvable f-description. The KB algorithm decides whether or not the f-description is solvable and, if so, constructs a minimal solution for it. Thus, the KB algorithm always halts for an arbitrary f-description. For details of the KB algorithm, see Kaplan & Bresnan (1982).

In the LFG theory, several *well-formedness conditions* are defined on f-structures such as *uniqueness*, *completeness*, and *coherency*. In this paper, we only consider the following uniqueness condition.

(*uniqueness*) In a given f-structure, each function name may have at most one value.

An f-structure satisfying this condition is said to be *well-formed* in this paper. Thus, if an f-description has a unique minimal solution, the solution is a well-formed f-structure because it satisfies·the uniqueness condition. We do not describe the other conditions here because they are not necessary in this paper. For details of the other conditions, see Kaplan & Bresnan (1982).

A string is *grammatical* only if it has a valid c-structure and it is assigned a well-formed f-structure. A language *generated by an LFG G*, denoted by $L(G)$, is a set of grammatical sentences of G. The class of languages generated by lexical functional grammars is denoted by L_{LFG}. Let *CFL* be the class of context-free languages. From the definition of LFG, it is obvious that the class of languages generated by LFGs includes *CFL*. On the other hand, it is easy to see that $L(G_1) = \{a^k b^k c^k \mid k \geq 1\}$. And it is well known that this language is not context-free. Thus we obtain $CFL \subsetneq L_{LFG}$. In fact, Kaplan and Bresnan have shown the following theorem.

Theorem 1 (Kaplan & Bresnan (1982)) $CFL \subsetneq L_{LFG} \subseteq$ *recursive sets* □

3. Computational Intractabilities of LFGs

In Berwick (1982), R. C. Berwick has shown that the membership problem for LFGs is NP-hard and mentioned that the LFG theory must be tightened if one wants to avoid computational intractability. In this section, we show some other results which also suggest the computational intractabilities of LFGs. They are as follows : (1) the emptiness problem for LFGs is undecidable, and (2) the membership problem for LFGs which have one ε-production is *EXPTIME*-hard.

The *emptiness problem* for LFGs is as follows : given an LFG G, decide whether $L(G)=\emptyset$. We first show that the emptiness problem for LFGs is undecidable. We reduce Post's Corresponding Problem to the emptiness problem. An instance of *Post's Corresponding Problem* (*PCP* for short) consists of two lists, $A = (x_1, x_2, \cdots, x_k)$ and $B = (y_1, y_2 \cdots, y_k)$ of strings over some alphabet Σ. An instance of PCP is said to *have a solution* if there exists a sequence of integers $i_1, i_2, \cdots, i_m$ $(m \geq 1)$ such that

$$x_{i1}, x_{i2}, \cdots, x_{im} = y_{i1}, y_{i2}, \cdots, y_{im}.$$

In this case, the sequence $i_1, i_2, \cdots, i_m$ is called a *solution* to this instance of PCP, and the string $x_{i1}, x_{i2}, \cdots, x_{im}$ $(= y_{i1}, y_{i2}, \cdots, y_{im})$ is called a *value* of this instance. The following theorem is well known.

Theorem 2 (Hopcroft & Ullman (1979), Moll et al. (1988)) *PCP is undecidable.* $\square$

Example 2 Let $\Sigma = \{0, 1\}$, $A = (01, 11)$ and $B = (1011, 1)$. This instance of PCP has a solution $i_1=2$, $i_2=1$ and $i_3=2$ (i. e. $m=3$). Then $x_2\ x_1\ x_2=y_2\ y_1\ y_2=110111$.

Theorem 3 (Nishino (1991)) *The emptiness problem for LFGs is undecidable.*

Proof Sketch. We reduce PCP to the emptiness problem. Let $\Sigma=\{0, 1\}$. And let $A=(x_1, x_2, \cdots, x_k)$ and $B=(y_1, y_2, \cdots, y_k)$ be an instance of PCP, where for each i $(1 \leq i \leq k)$, $x_i=a_1\ a_2 \cdots a_{m(i)}$ with $a_j \in \Sigma$ for each j $(1 \leq j \leq m(i))$, and $y_i=b_1\ b_2 \cdots b_{n(i)}$ with $b_j \in \Sigma$ for each j $(1 \leq j \leq n(i))$. From the lists A and B, we construct an LFG $G(A, B)$ such that an instance (A, B) of PCP has a value y (i. e. has a solution) iff $(y^R)^3 \in L(G(A, B))$ iff $L(G$

$(A, B)) \neq \emptyset$, where y^R denotes the reverse of y. For details, see Nishino (1991).□

Next, we show that the membership problem for LFGs which have one ε-production is *EXPTIME*-hard. We first briefly describe the basic concepts in computational complexity: For details, see Hopcroft & Ullman (1979). Let x be a string over some alphabet. We denote the length of x by $|x|$. Let n be the length of an input to a Turing machine. $DTIME(T(n))$ is the class of languages accepted by deterministic Turing machines within $T(n)$ time. We define $EXPTIME = \bigcup_{i \geq 0} DTIME(2^{n^i})$. For a Turing machine M, we denote by $L(M)$ the language accepted by M.

Let L_1 and L_2 be languages (i. e. sets of strings). L_1 is said to be *polynomial-time reducible* to L_2 if there is a polynomial-time bounded deterministic Turing machine that for each input x produces an output y such that $y \in L_2$ if $x \in L_1$. A language L is said to be *EXPTIME-hard* if every language in *EXPTIME* is polynomial-time reducible to L.

Definition 2 A *one-tape alternating Turing machine* (ATM for short) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, U)$ where :

1. Q is the finite set of *states.*
2. Σ is the finite *input alphabet.*
3. Γ is the finite *tape alphabet.*
4. δ is the *next move relation* mapping an element of $Q \times \Sigma$ to a subset of $Q \times \Sigma \times D$, where $D = \{L, R\}$.
5. $q_0 \in Q$ is the *initial state.*
6. $q_a \in Q$ is the *accepting state.*
7. $U \subseteq Q$ is the set of *universal states.* $Q - U$ is called the set of *existential states.*

A machine move is represented as follows. Let $\delta(q, x)$ be of the following form.

$$\delta(q, x) = \{(q_1, y_1, d_1), (q_2, y_2, d_2), \cdots, (q_m, y_m, d_m)\},$$

where $q, q_1, \cdots, q_m \in Q$, $d_1, d_2, \cdots, d_m \in D$ and $x, y_1, \cdots, y_m \in \Sigma$. In state q, scanning symbol x, M takes the following action ACT(i) for some i, $1 \leq i \leq m$ if q is an existential state, and takes ACT(i) for all i, $1 \leq i \leq m$ if q is a universal state.

ACT(i) : Rewrite x as y_i, move the tape head one position in the direction of d_i, and change the state to q_i.

A *configuration* of M consists of the state, head position, and contents of the tape. Let C be a configuration of M. We denote the set of possible configurations after one move of M by $Next(C)$. A configuration is *existential* (resp. *universal, initial, accepting*) if the state of M in that configuration is an existential (resp. universal, initial, accepting) state. A value $v(C)$ of C is either true or false defined by the procedure shown in Fig. 4. An ATM accepts an input string x if $v(C_0) = true$, where C_0 is the initial configuration for x. Notice that, without losing generality, we can assume that an ATM has only one accepting state.

```
Procedure  EVAL ($C$ : a configuration of an ATM) ;
    begin
      if $C$ is an accepting configuration
          then $U(C) := true$
          else if C is an existential configuration
            then if there is $C' \in Next(C)$ such that $v(C') = true$
                then $v(C) := true$
                else $v(C) := false$
            else   % $C$ is a universal configuration. %
                if for every configuration $C' \in Next(C)$, $v(C') = true$
                    then $v(C) := true$
                    else $v(C) := false$
    end
```

Figure 4. The procedure to define the value $v(C)$ of C.

$ASPACE(S(n))$ is the class of languages accepted by ATMs within $S(n)$ space. The following theorem states that the time complexity of the recognition problem for linear space-bounded ATMs is exponential in terms of deterministic Turing machine.

Theorem 4 (Chandra et al. (1981)) $EXPTIME = \bigcup_{i \geq 0} ASPACE(n^i)$ □

Theorem 5 (Nishino (1991)) *The membership problem for LFGs which have one ε-production is EXPTIME-hard.*

Proof Sketch. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, U)$ be a one-tape linear space ATM. We assume that the length of the tape is exactly n. Let $w = x_1 x_2 \cdots x_n$ be an input for M, where $x_i \in \Sigma$ for all i, $1 \leq i \leq n$. We will construct an LFG $G(M, \omega) = (\Sigma, \Gamma, S, FN, FV, AR)$ having one ε-production, such that M accepts ω iff $\omega \in L(D(M, \omega))$. For details, see Nishino (1991). □

We summarize the results of this section in Table 1.

	CFG	LFG	*CSG*
Emptiness	*PTIME*[6]	undecidable	*undecidable [6]*
Membership	*PTIME*[6]	*EXPTIM hard**	*NSPACE*(n) [6]

Remarks * In the case when the LFG has one ε-production.

Table 1. Summary of the results of this section.

4. Frontier-to-Root Lexical-Functional Grammars

Based on the above observations, we propose a new subclass of LFGs meeting the dual demands of parsability and learnability from computational theoretic points of view. That is, we introduce *Frontier-to-Root LFGs* (FRLFGs for short) by restricting the form of functional equations attached to phrase structure rules.

In a sentence description of FRLFGs, there are two types of components, a c-structure, which is the same with LFGs, and a *functional tree* (*f-tree*). An f-tree is a rooted ordered tree whose internal nodes are labeled by function names and leafs are labeled by function values. Using these two structures, and FRLFG system specify a set of grammatical sentences. The following definition is a restricted version of Definition 1.

Definition 3 A *frontier-to-root lexical functional grammar* (FRLFG for short) is a 6-tuple $G = (NA, TA, S, FN, FV, AR)$ consists of 1-6 as follows :

1. NA is *nonterminal alphabet.*
2. TA is a *terminal alphabet.*
3. S is a *start symbol.*
4. FN is a finite set of *function names.*

5. FV is a finite set of *function values.*
6. AR is a finite set of *annotated phrase structure rules.* An annotated phrase structure rule is of the form

$$A \rightarrow (B_1, E_1)\ (B_2, E_2) \cdots (B_m, E_m),$$

where $m \geq 0$, $A \in NA$, $B_i \in NA \cup TA$ for each i, $1 \leq i \leq m$ and E_i $(1 \leq i \leq m)$ is a set of functional assignments. We assume that at least one B_i is a nonterminal symbol. A *functional assignment* is a statement of one of the following forms :

$$((\uparrow F_1) F_2) := \downarrow, \qquad (\uparrow F_1) := \downarrow, \qquad \uparrow := \downarrow,$$
$$((\uparrow F_1) F_2) := V, \qquad (\uparrow F_1) := V, \qquad \uparrow := V,$$

where F_1 and F_2 are function names and V is a function value. The symbols $\uparrow$ and $\downarrow$ are called *metavariables.* An annotated phrase structure rule of the form $A \rightarrow (b, E)$ is especially called a *lexical insertion rule,* where $b \in TA$ and E is a functional assignment set such that the righthand side of each rule in E is a function value. *Each functional assignment set is assumed to be a singleton (i. e. only one assignment is included) apart from the sets attached to lexical insertion rules.*

Example 3 Let us consider the following simple FRLFG.
$G_2 = (NA, TA, S, FN, FV, AR)$, where
$NA = \{S, A, B, C\}$, $TA = \{a, b, c\}$, $FN = \{K\}$, $FV = \{0\}$, and
AR involves the following rules :
$S \rightarrow (A, \{\uparrow := \downarrow\})\ (B, \{\uparrow := \downarrow\})\ (C, \{\uparrow := \downarrow\})$,
$A \rightarrow (a, \emptyset)\ (A, \{(\uparrow K) := \downarrow\})$,
$B \rightarrow (b, \emptyset)\ (B, \{(\uparrow K) := \downarrow\})$,
$C \rightarrow (c, \emptyset)\ (C, \{(\uparrow K) := \downarrow\})$,
$A \rightarrow (a, \{(\uparrow K) := 0\})$,
$B \rightarrow (b, \{(\uparrow K) := 0\})$, and
$C \rightarrow (c, \{(\uparrow K) := 0\})$.
The underlying CFG of G_2 is equal to the CFG Gr_1 in Example 1.

Fig. 5 (a) illustrates an example of a c-structue generated by the above FRLFG G_2. While an f-tree is a rooted ordered tree whose internal nodes labeled by function names and leaves are labeled by their values as shown in Fig. 5 (b).

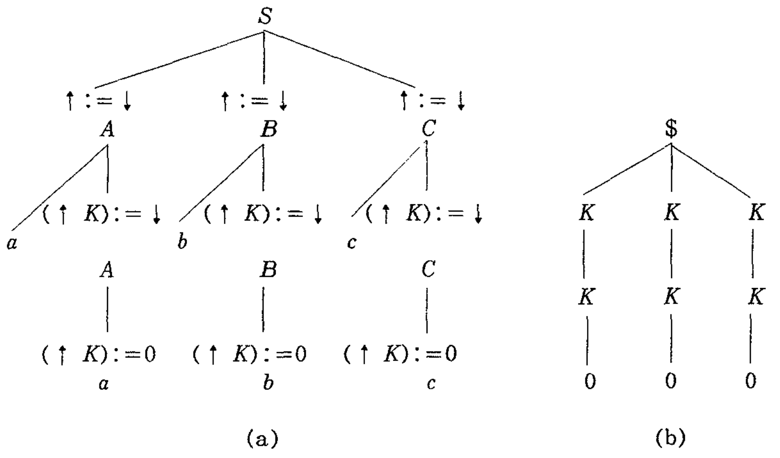

Figure 5. A derivation of the string $x = aabbcc$ in G_2, (a) an annotated phrase structure tree for x, (b) a well-formed f-tree $f(x)$ assigned to x.

Each node in a c-structure is assumed to have an associated f-tree. The metavariable ↓ attached to a node n of a c-structure represents the f-tree associated to n. While the metavariable ↑ attached to n represents the f-tree associated to the parent of n. In this situation, a functional assignment of the form ↑ : = ↓ attached to n represents that the f-tree associated to n is *concatenated* to the f-tree associated to the parent of n. (A definition of the concatenation is given in later.) On the other hand, roughly speaking, a functional assignment of the form (↑ K) : = ↓ attached to n represents that the f-tree associated to n is concatenated to a node labeled by K in the f-tree associated to the parent of n.

Given a c-structure t for a string x, the f-tree assigned to x, denoted $f(x)$, is synthesized in the following way. Traversing t in the *depth-first left-to-right order*, functional assignments are evaluated at each node. Let n be a node in t labeled by A and expanded by the following annotated phrase structure rule

$$p : A \rightarrow (B_1, E_1)\ (B_2, E_2) \cdots (B_m, E_m).$$

The f-tree associated to n (represented by ↑ metavariable appearing in E_i, $1 \leq i \leq m$) is synthesized by performing all tree concatenations expressed by assignments in $\bigcup_{1 \leq i \leq m} E_i$. We assume that before $\uparrow$ is evaluated, all the values of $\downarrow$ metavariables appearing in $\bigcup_{1 \leq i \leq m} E_i$ have already been evaluated. An initial value of $\uparrow$ is a tree consisting of only one node labeled by \$. First, the functional assignments in E_1 are evaluated in an arbitrary order. (Notice that if p is not a lexical insertion rule, there is only one assignment in E_1 by definition.) When these evaluations are finished, then the functional assignments in E_2 are evaluated, and so on. Let $e \in E_i$ be a functional assignment of the following form :

$$((\uparrow F_1) F_2) := \downarrow .$$

This assignment is processed as follows : a node labeled by F_1 is firstly concatenated to the \$ -node as the rightmost son of it, and then a node labeled by F_2 is concatenated to the F_1-node as a unique son. Then a tree t_i, which is a value of $\downarrow$, is concatenated to the F_2-node as shown in Fig. 6. Other types of functional assignments are similarly processed.

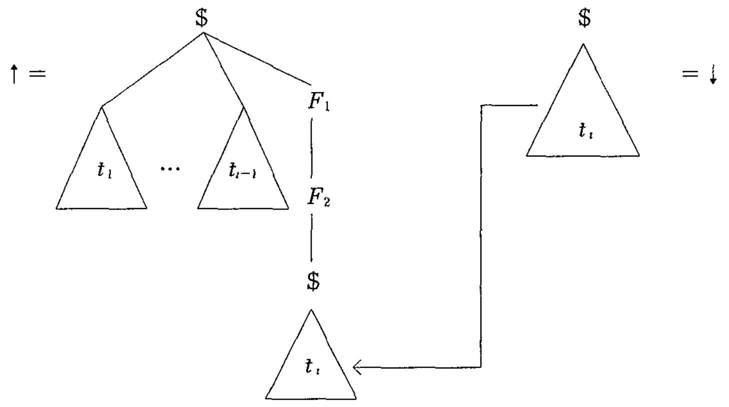

Figure 6. A tree concatenation operation expressed by the functional assignment $((\uparrow F_1) F_2) := \downarrow$.

The *size* of an f-tree is the number of nodes in it. From the forms of functional assignments, it is obvious that the size of the f-tree associated to a node n is always smaller than or equal to that of the parent of n. Thus, in a c-structure t, the root of t is associated with the largest f-tree in t. This largest f-tree is called the f-tree *assigned* to the string x and denoted by $f(x)$. Fig. 5 (b) illustrates an f-tree assigned to the string *aabbcc* in G_2.

Notice that in FRLFGs, since we can obtain f-trees directly from an annotated phrase structure trees, we do not need f-descriptions. Furthermore, from an annotated phrase structure tree t, the corresponding f-tree is uniquely determined since the order of traversing in t is defined to be the depth-first left-to-right order.

In the FRLFG theory, several *well formedness conditions* are defined on f-trees. A well-formedness condition is an arbitrary condition on f-trees which can be checked within $O(|f|^2)$ time, and within $O(|f|)$ space, where $|f|$ is the size of f. In this paper, we assume that an f-tree f is said to be *well-formed* if all the subtrees of f, whose roots are sons of the root of f, are identical. For example, the f-tree $f(x)$ in Fig. 5 (b) is well-formed in this sense. Notice that this condition, in fact, can be checked within $O(|f|)$ time and $O(|f|)$ space for an arbitrary f-tree f. In linguistic arguments, the well-formedness conditions are more complicated.

A string $x \in TA^*$ is *grammatical* only if it has a valid c-structure and it is assigned a well-formed f-tree $f(x)$. A language *generated by an FRLFG* G, denoted by $L(G)$, is a set of grammatical strings of G. Notice that if the underlying CFG of G is ambiguous, in order to decide whether $x \in L(G)$, we need to check the well-formedness of the corresponding f-tree for each c-structure of x in general. It is easy to see that $L(G_2) = \{a^k b^k c^k \mid k \geq 1\}$. The class of languages generated by FRLFGs is denoted by L_{FRLFG}. Let CSL be the class of context-sensitive languages. We obtain the following theorem.

Theorem 6 (Nishino (1991)) $CFL \subsetneq L_{FRLFG} \subseteq CSL$ □

5. Parsability and Learnability of FRLFG Languages

In this section, we briefly describe some formal properties of FRLFGs. The reader is referred to Nishino (1991) for the proofs of the theorems presented in this section.

5. 1. Parsing FRLFG Languages

Let G be a λ-free CFG having no $X_i \rightarrow X_j$ type productions. For all $x \in TA^*$ we define the integer $d(x, G) \geq 0$ to be the *degree of ambiguity of* x *with respect to* G, which is the number of the different parsing trees of x. If

$x \notin L(G)$ then $d(x, G)=0$. It is known that, for any x and G, $d(x, G)$ is decidable using formal power series (Révész (1983)). Let $|x|=n$. Notice that $d(x, G)$ may be a function of n. Concerning the problem of parsing FRLFG languages, we obtain the following theorems.

Theorem 7 (Nishino (1991)) *Let G be an FRLFG, and $x \in TA^*$ with $|x| = n$. Then there is an algorithm deciding whether $x \in L(G)$ and if so, generating all the different c-structures and f-trees of x in $O(n^3+d(x, Gr) \cdot n)$ time, where Gr is the underlying CFG of G.* □

Theorem 8 (Nishino (1991)) *Let G be an FRLFG such that the underlying CFG Gr is unambiguous. And let $x \in TA^*$ with $|x|=n$. Then there is an algorithm deciding whether $x \in L(G)$ and if so, generating the unique c-structure and f-tree of x in $O(n^2)$ time.* □

Example 4 For a string $x \in L(G_2)$ of length n, the size of the f-tree $f(x)$ assigned to x is at most $n+4$. Since G_2 is unambiguous, we obtain by Theorem 8 that the membership and parsing problems for the language $\{a^k b^k c^k \mid k \geq 1\}$ can be solved in $O(n^2)$ time using the FRLFG framework.

Let us consider an another FRLFG $G_3 = (NA, TA, S, FN, FV, AR)$ such that

$NA = \{S, W, X\}$, $TA = \{a, b\}$, $FN = \{W, X\}$, $FV = \{A, B\}$, and

AR involves the following rules :

$S \rightarrow (W, \{\uparrow := \downarrow\})\ (W, \{\uparrow := \downarrow\})$,

$W \rightarrow (X, \{\uparrow := \downarrow\})\ (W, \{(\uparrow W) := \downarrow\})$, $W \rightarrow (X, \{\uparrow := \downarrow\})$,

$X \rightarrow (a, \{(\uparrow X) := A\})$, and $X \rightarrow (b, \{(\uparrow X) := B\})$.

It is easy to see that $L(G_3) = \{ww \mid w \in TA^*\}$ and that, for a string $x \in L(G_3)$ of length n, the size of the f-tree $f(x)$ assigned to x is at most $3n-1$. This grammar G_3 is ambiguous and, for a string x of length n, $d(x, G_3) = n-1$. Thus we obtain by Theorem 7 that the membership and parsing problems for the language $\{ww \mid w \in \Sigma^*\}$ can be solved in $O(n^3)$ time using the FRLFG framework.

5.2. A Characterization of FRLFG Translations

Next, we show that a translation realized by an FRLFG can be defined in terms of a tree transduction realized by a frontier-to-root tree automaton with output. For details of tree transducers, see Gécseg & Steinby (1984), Thatcher (1973).

We first briefly review the definitions of trees and tree transducers. An empty string is denoted by λ and the power set of a set A is denoted by 2^A. For a set A, A^* denotes the set of all finite strings over A. For $x \in A^*$, the length of x is denoted by $|x|$. Let N be the set of non-negative integers, and $N_+ = N - \{0\}$. For $x, y \in N_+^*$, $x \geq y$ if there exists $z \in N_+^*$ such that $x = yz$, and $x > y$ if $x \geq y$ and $x \neq y$.

A *ranked alphabet* is defined to be a pair $V = <\Sigma, \sigma>$, where Σ is a finite set of symbols and σ is a mapping from Σ into N. For $a \in \Sigma$, $\sigma(a)$ is called the *rank* of a. We will denote the set of symbols of rank n by Σ_n, i.e. $\Sigma_n = \sigma^{-1}(n)$. A symbol in the set Σ_0 is called a *constant symbol.*

Definition 4 A *Σ-tree*, or a *tree over Σ* is a mapping t from $Dom(t)$ into Σ, where $Dom(t)$ is a finite subset of N_+^* satisfying the following conditions :

1. If $x \in Dom(t)$ and $x > y$ then $y \in Dom(t)$.
2. If $yi \in Dom(t)$ for $i \in N_+$ then $yj \in Dom(t)$ for $j \in N_+$, $1 \leq j \leq i$.
3. If $t(x) = a \in \Sigma_n$ then $xi \in Dom(t)$ for $i \in N_+$, $1 \leq i \leq n$.

An element of $Dom(t)$ is called a *node* of t. If $t(x) = a$, then a is said to be the *label* of the node x of t. The set of all trees over Σ is denoted by T_Σ.

A *skeletal alphabet* $\Sigma = \bigcup_{i=0}^{n} \Sigma_i$ with the maximal rank n is a ranked alphabet such that $\Sigma_i = \{\sigma\}$ for each i, $1 \leq i \leq n$, where σ is a special symbol. A tree over a skeletal alphabet is called a *skeleton.* The *structural description* of a tree t, which is denoted by $s(t)$, is a skeleton with $Dom(s(t)) = Dom(t)$ such that if x is a terminal node in $Dom(t)$ then $s(t)(x) = t(x)$ else $s(t)(x) = \sigma$.

Definition 5 A *deterministic frontier-to-root skeletal automaton with output* (FRSAO for short) is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, \gamma, F)$ consists of 1-6 as follows:

1. Q is a finite set of *states.*
2. Σ is an input *skeletal alphabet* with the maximal rank n.
3. Γ is an output ranked alphabet with the maximal rank m. Γ includes special symbols $x_1, \cdots, x_n$.
4. $\delta = (\delta_1, \delta_2, \cdots, \delta_n)$ is the *state transition function,* where

$\delta_k : \Sigma_k \times Q^k \to Q \qquad (k = 0, 1, \cdots, n)$,

5. $\gamma = (\gamma_0, \gamma_1, \cdots, \gamma_n)$ is the *output function*, where
$\gamma_k : \Sigma_k \times Q^k \to T_\Gamma \ (k = 1, \cdots, n)$, $\gamma_0 : \Sigma_0 \to T_{\Gamma - \{x_1, \cdots, x_n\}}$, and

6. $F \subseteq Q$ is the set of *final states.*

We define the set of skeletons *accepted* by M as $L(M) = \{t \in T_\Sigma \mid \delta(t) \in F\}$. On the other hand, an FRSAO realizes a *translation* T_M from the input skeletons over Σ to the output trees over Γ. That is, for an input skeleton $t \in T_\Sigma$, M generates a tree over Γ, which is denoted $T_M(t)$. Before we describe how to generate an output tree, we need some terminologies.

We denote by $t \leftarrow [t_1, \cdots, t_k]$ a tree obtained by substituting t_i for x_i simultaneously in t, for each i, $1 \leq i \leq k$. For example, let us consider a subtree t of an input tree shown in Fig. 7 (a), where the label of the root is σ, and $q_i = \delta(t_i)$ for each i, $1 \leq i \leq k$. Suppose an FRSAO M has a Γ-tree shown in Fig. 7 (b) as a value of the output function $\gamma_k(\sigma, q_1, \cdots, q_k)$. Then the output tree corresponding to t, which is denoted $T_M(t)$, is the tree in Fig. 7(c), where $t'_i = T_M(t_i)$ for each i, $1 \leq i \leq k$.

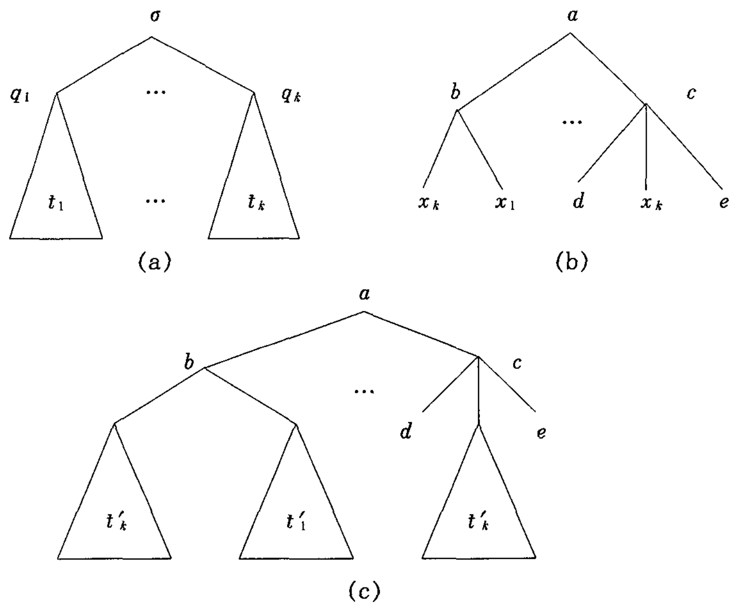

Figure 7. Synthesis of an output tree by an FRSAO, (a) a subtree t of an input tree, (b) a Γ-tree $\gamma_k(\sigma, q_1, \cdots, q_k)$, (c) a Γ-tree $\gamma_k(\sigma, q_1, \cdots, q_k) \leftarrow [t'_1, \cdots, t'_k]$.

An FRSAO M induces a following mapping $T_M : T_\Sigma \rightarrow Tr$ such that

1. If t is a tree consisting of only one node labeled by $a \in \Sigma_0$, then $T_M(t) = \gamma_0(a)$.
2. In general, if t is of the form shown in Fig. 7 then

$$T_M(t) = \gamma_k(\sigma, q_1, \cdots, q_k) \leftarrow [T_M(t_1), \cdots, T_M(t_k)],$$

where $\delta(t_i) = q_i$, for i, $1 \leq i \leq k$.

The *output set* of M is defined as $OUT(M) = \{T_M(t) \mid t \in L(M)\}$.

Let t be a tree. A *path* in t is a sequence $n_1\, n_2 \cdots n_k$ of nodes in t such that

1. n_1 is a root of t,
2. $n_2, \cdots, n_{k-1}$ are all internal nodes, but none of them is a root,
3. n_k is a leaf of t, and
4. n_{i+1} is a son of n_i in t for each i, $1 \leq i \leq k-1$.

Let $p = n_1\, n_2 \cdots n_k$ be a path in t such that a label of n_i is l_i $(1 \leq i \leq k)$. The sequence of labels $l_1\, l_2 \cdots l_k$ is denoted $LAB(p)$. We define a new operation *ERASE* as follows :

$$ERASE(l_1\, l_2 \cdots l_k) = l_1\, l'_2 \cdots l'_k$$

where, for each i, $2 \leq i \leq k$, if $l_i = \$$ then $l'_i = \lambda$ *else*$= l'_i = l_i$. For a tree t, a set $\Pi(t)$ is defined as follows :

$$\Pi(t) = \{ERASE(LAB(p)) \mid p \text{ is a path in } t\}.$$

Furthermore, for a set of trees L, we define

$$\Pi(L) = \{\Pi(t) \mid t \in L\}.$$

Let G be an FRLFG. A *translation* realized by G, denoted by T_G, is defined as follows. $T_G : TA^* \rightarrow T_{FV \cup FN}$, $T_G(x) = f(x)$ for $x \in TA^*$. Notice that T_G is a partial function. The output set of G is defined as $OUT(G) = \{T_G(x) \mid x \in L(G)\}$. We obtain the following theorem.

Theorem 9 (Nishino (1991)) *Let G be an FRLFG $G = (NA, TA, S, FN, FV, AR)$. Then there exists an FRSAO $M = (Q, \Sigma, \Gamma, \delta, \gamma, F)$ such that $\Sigma_0 = TA$, $\Gamma = FN \cup FV \cup \{\$, x_1, \cdots, x_n\}$ for some constant n, and $\Pi(T_G(x)) = \Pi(T_M(s(t)))$ for all c-structure t of G, where x is the yield of t.* □

5.3. Learning FRLFGs

Let G_U be an unknown FRLFG, and L_U be the context-free language generated by the underlying CFG of G_U. And let M_U be the minimum FRSAO accepting L_U and generating $OUT(G_U)$. It is assumed that the learner knows the ranked alphabets Σ and Γ of M_U. The learner can propose the following queries to the teacher :

(1) A *structural membership query* proposes a skeleton t and asks whether $t \in L(M_U)$. The answer from the teacher is either *yes* or *no*.

(2) A *structural equivalence query* proposes a FRSAO M and asks whether $L(M) = L(M_U)$. The answer is either *yes* or *no*. When the answer is *no*, a *counterexample* is also provided from the teacher. It is a skeleon t in the symmetric difference of $L(M_U)$ and $L(M)$.

(3) A *structural output query* proposes a skeleton t and asks $\gamma_U(t)$, where γ_U is the output function of M_U. The answer from the teacher is this output $\gamma_U(t) \in T_\Gamma$

This learning protocol is based on Angluin's "minimally adequate teacher" in Angluin (1987) and is illustrated in Fig. 8. Our algorithm is an extension of Sakakibara's learning algorithm for deterministic tree acceptors in Sakakibara (1988) to the one for deterministic tree transducers. Our learning algorithm consists of the following two steps.

Step 1. Learning of the underlying CFG of G_U.

Step 2. Learning of the functional assignments of G_U.

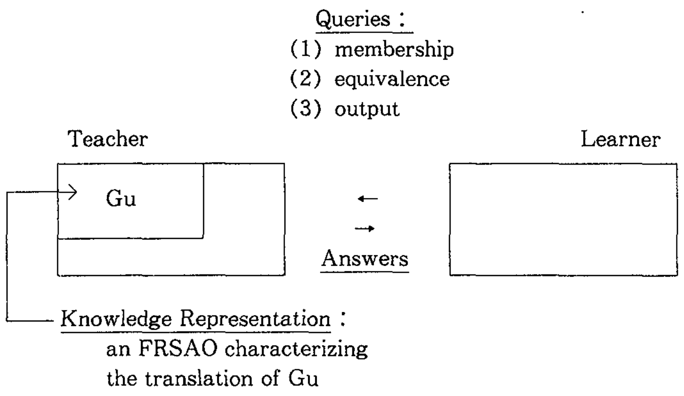
Queries :
(1) membership
(2) equivalence
(3) output
Teacher
Learner
Gu
Answers
Knowledge Representation :
an FRSAO characterizing the translation of Gu

Figure 8. The learning protocol.

In Step 1, we use the Sakakibara's learning algorithm. For details, see Sakakibara (1988). Using a well known coding of the recognizing process of a tree automation in the formalism of a CFG, Sakakibara has shown the following result.

Theorem 10 (Sakakibara (1988)) *Let G_U be an unknown CFG. There is an algorithm that learns a CFG structurally equivalent to G_U using structural membership and structural equivalence queries that runs in time polynomial in the number of states of the minimum deterministic frontier-to-root skeletal automaton for the structural description of G, and the maximum size of any counterexample provided by the teacher.* $\square$

In Nishino (1991), it is shown that, based on the results of Step 1, one can learn the unknown functional assignments of the target FRLFG G_U using structural output queries. That is, we obtain the following theorem.

Theorem 11 (Nishino (1991)) *Let G_U be an unknown FRLFG, and L_U be the context-free language generated by the underlying CFG of G_U. And let M_U be the minimum FRSAO accepting the structural description of L_U and generating $OUT(G_U)$. Then there is an algorithm to learn the set of annotated phrase structure rules of G_U using structural membership, structural equivalence, and structural output queries for L_U. Moreover, this algorithm runs in time polynomial in the number of the states of M_U, and the maximum size of any counterexample provided by the teacher.* $\square$

6. Concluding Remarks

In most natural language applications such as machine translation and question answering, it is needed to select a set of plausible parsing trees from among all the alternatives. The cost of recovering an individual parsing tree from a parsing table depends on the degree of ambiguity. Thus, this process may be very time-consuming even if we use an efficient parsing algorithm for context-free languages such as the CKY or Early's algorithm. Based on this observation, by Theorem 7, we may say that FRLFGs are nearly optimal grammar beyond context-free.

It seems that a large part of linguistic examples in the literature can be described using FRLFGs. It is very important to investigate various linguistic problems within this framework, in order to discuss whether FRLFG is a realistic model of lexical-functional grammar from a linguistic points of view.

It is another interesting subject to implement a learning program for context-sensitive languages based on the results of Section 5. This is a subject for further research.

Acknowledgments

The author would like to express his sincere thanks to Professor Hiroshi Noguchi of Waseda University for his kind advice. He also thanks Professor Joan Bresnan of Stanford University for her valuable suggestions.

References

Angluin, D. (1987) 'Learning Regular Sets from Queries and Counterexamples,' *Information and Computation*, Vol. 75, pp. 87-106.

Berwick, R. C. (1982) 'Computational Complexity and Lexical Functional Grammar,' *American Journal of Computational Linguistics*, 8(3-4), pp. 97-109.

Bresnan, J. (ed.) (1982) *The Mental Representation of Grammatical Relations*, MIT Press, Camblidge, Massachusetts.

Chandra, A. K., D. C. Kozen and L. J. Stockmeyer (1981) 'Alternation,' *JACM*, 28, pp. 114-133.

Gēcseg, F. and M. Steinby (1984) *Tree Automata*, Akadēmiai Kiadō, Budapest.

Hopcroft, J. E. and J. D. Ullman (1979) *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley.

Kaplan, R. M. and J. Bresnan (1982) 'Lexical-Functional Grammar : A Formal System for Grammatical Representation,' In : J. Bresnan (ed.) *The Mental Representation of Grammatical Relations*, MIT Press, Camblidge, Massachusetts.

Moll, R. N., M. A. Arbib and A. J. Kfoury (1988) *An Introduction to Formal*

Language Theory, Springer-Verlag.

Nishino, T. (1991) *Formal Methods in Natural Language Syntax*, Doctoral dissertation, Waseda University.

Pinker, S. (1982) 'A Theory of the Acquisition of Lexical Interpretive Grammars,' In : J. Bresnan (ed.) *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts.

Pinker, S. (1984) *Language Learnability and Language Development*, Harvard University Press.

Révész, G. E. (1983) *Introduction to Formal Languages*, McGraw-Hill Book Company.

Sakakibara, Y. (1988) 'Learning Context-Free Grammars from Structural Data in Polynomial Time,' in *Proc. of the First Workshop on Computational Learning Theory*, Morgan Kaufmann.

Savitch, W. J. et al. (eds.) (1987) *The Formal Complexity of Natural Language*, D. Reidel Publishing Company.

Sells, P. (1985) *Lectures on Contemporary Syntactic Theories*, Center for the Study of Language and Information, Stanford University.

Thatcher, J. W. (1973) 'Tree Automata: An Informal Survey,' in Aho, A. V. (ed.), *Currents in the Theory of Computing*, Prentice-Hall.

Winograd, T. (1983) *Language as a Cognitive Process, Volume* I : *Syntax*, Addison-Wesley.

Department of Information Science
Tokyo Denki University
Hatoyama-Machi, Hiki-Gun, Saitama 350-03
Japan