数理解析研究所講究録
第790巻 1992年 43-49

根方向型語彙機能文法に基づく構文解析プログラムの実現

An Efficient Parser Based on Frontier-to-Root Lexical-Functional Grammars

山田節夫，西野哲朗，米田信夫
Setsuo Yamada, Tetsuro Nishino, and Nobuo Yoneda

東京電機大学 理工学部 情報科学科
Department of Information Sciences, Tokyo Denki University

Abstract

In 1990, frontier-to-root lexical-functional grammars (FRLFGs for short) were intoroduced to specify the human language syntax. 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. In this paper, extending the Earley's algorithm, we design an efficient parsing algorithm for FRLFG languages.

1 Introduction

In 1965, N. Chomsky introduced transformational grammars (TGs for short) to describe the syntax of natural languages. It was shown, however, that TGs can generate any recursively enumerable sets. From this fact, it seems that the generative power of TGs is too strong. Thus, many grammars whose generative power is weaker than that of TGs have been proposed. In 1982, extending context-free grammars, J. Bresnan and R. M. Kaplan introduced lexical-fuctional grammars (LFGs for short) as one of such kinds of grammars. But it is shown by R. C. Berwick that the membership problem for LFGs is NP-hard. Hence, it seems that LFGs are too complex to deal with on computers. Furthermore, some other results showing computational intractabilities of LFGs have been proved [5, 8]. So, in 1990, restricting LFGs, T. Nishino introduced frontier-to-root LFGs (FRLFGs for short) [6]. 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 [7].

In this paper, we show an efficient parsing algorithm for FRLFG languages, which consists of the following three algorithms :

(1) An extended Earley's algorithm;

(2) An f-tree construction algorithm; and

(3) A well-formedness checking algorithm.

The algorithm (1) is used to parse context-free languages. Then, the algorithm (2) constructs f-trees using the results of the algorithm (1) and functional assignments attached to context-free rules. Finally, the algorithm (3) tests whether the f-trees constructed by the algorithm (2) are well-formed.

2 Frontier-to-Root Lexical-Functional Grammars

In this section, we describe the definition of FRLFGs. For details, see [6, 7]. For details of formal laguages and trees, see [3] for example.

2.1 A Formal Definition

Definition 2.1 A *frontier-to-root lexical-fuctional grammar* (FRLFG for short) G is a 6-tuple (NA, TA, S, FN, FV, AR) consists of 1-6 as follows :

1. NA is a *nonterminal alphabet.*
2. TA is a *terminal alphabet.*
3. $S \in NA$ 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.*

We assume that $NA \cap TA = \emptyset$ and $FN \cap FV = \emptyset$. An annotated phrase structure rule is of the form

$$A \rightarrow (B_1, E_1)(B_2, E_2) \cdots (B_n, E_n),$$

where $n \geq 1$, $A \in NA$, and $B_i \in NA \cup TA$ $(1 \leq i \leq n)$. If $n \geq 2$, we assume that at least one of $B_1, B_2, \ldots, B_n$ is a nonterminal symbol. If $n = 1$, B_1 may be an empty string ε. Each E_i is a set of *functional assignments.* A functional assignment is a statement of one of the following forms :

(1) (in the case when $B_i \in NA$)

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

(2) (in the case when $B_i \in TA \cup \{\varepsilon\}$)

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

where $F_1, F_2 \in FN$ and $V \in FV$. The symbols $\uparrow$ and $\downarrow$ are called *metavariables.* Especially, annotated phrase structure rules of the following forms,

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

are called a *lexical insertion rule* and an *ε-rule* respectively, where $b \in TA$ and E is a nonempty finite set of functional assignments of the forms in (2). We assume that *each set of functional assignments is a singleton except the sets attached to the lexical insertion rules and the ε-rules.*

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

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

We call a derivation tree of an underlying CFG of an FRLFG a *constituent tree* (*c-tree*). A tree which is obtained by attaching functional assignments to a c-tree is called an *annotated phrase structure tree.* We assume that *the underlying CFG is cycle-free.*

Now we define f-trees. A *functional tree* (*f-tree*) f is a rooted ordered tree which satisfies the followings :

1. The root of f is labeled by a special symbol \$.
2. Each internal node in f apart from the root is labeled by \$ or a function name.
3. The leaves of f are labeled by function values.

We assume that each internal node in an annotated phrase structure tree is associated with an f-tree. A metavariable $\downarrow$ and $\uparrow$ are interpreted as follows :

The metavariable $\downarrow$ attached to a node n represents the f-tree associated to n; and

The metavariable $\uparrow$ attached to n represents the f-tree associated to the father of n.

For example, a functional assignment of the form $\uparrow := \downarrow$ attached to n represents that the f-tree associated to n is *concatenated* to the f-tree associated to the father of n.

Now, we describe how to synthesize the f-tree *assigned* to a string x. This f-tree is denoted by $f(x)$, and associated to the root of an annotated phrase structure tree t for a string x. Traversing t in *depth-first left-to-right order*, functional assignments are evaluated at each node. Let X_0 be a node in t labeled by A and expanded by an annotated phrase structure rule of the form

$$p \ : \ A \rightarrow (B_1, E_1)(B_2, E_2) \cdots (B_n, E_n).$$

The f-tree associated to X_0 (represented by $\uparrow$ metavariables appearing in E_i, $1 \leq i \leq n$) is synthesized by performing all tree concatenations expressed by functional assignments in $\bigcup_{i=1}^{n} E_i$. Because t is traversed in depth-first left-to-right order, we can assume that before $\uparrow$ is evaluated, all values of $\downarrow$ appearing in $\bigcup_{i=1}^{n} E_i$ have already been evaluated. An initial value of $\uparrow$ is a tree which consists of only one node labeled by \$. If p is a lexical insertion rule or an ε-rule, the functional assignments in E_1 can be evaluated in any order. Otherwise, the only one functional assignment in E_1 is firstly evaluated. Secondly, the only one functional assignment in E_2 is evaluated. And this process is repeated until the functional assignment in E_n is evaluated.

2.2 A Membership Procedure for the FRLFG Languages

In Fig. 1, we show a procedure for the FRLFG membership. Here, we define *well-formedness conditions* for an f-tree $f(x)$ as the following three conditions :

1. *uniqueness*,
2. *completeness*, and

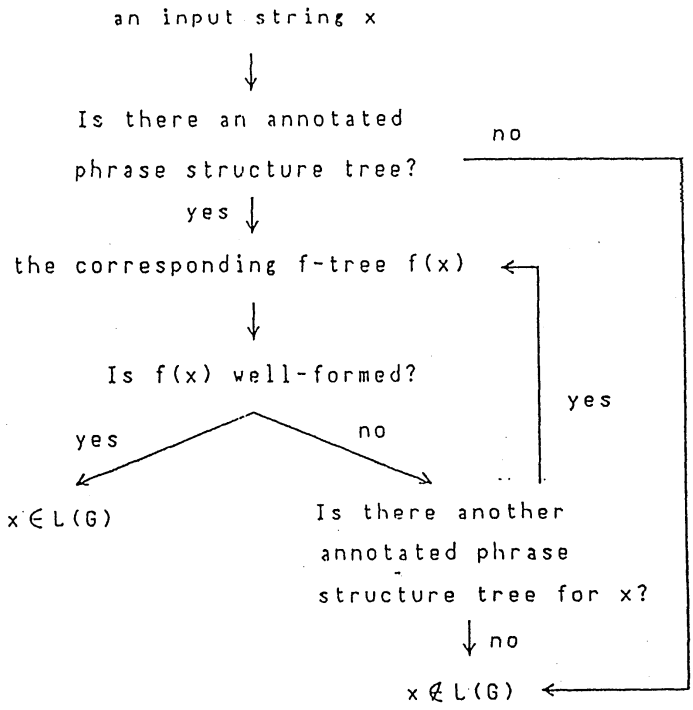

Figure 1: A procedure for the FRLFG membership.

3. *coherency.*

In order to define these three conditions, we need some terminologies. A *\$-free-tree corresponding to an f-tree f* is an f-tree obtained by removing all internal nodes of f, excepting the root of f, whose labeles are \$. A node n is a *pre-terminal node* if n is an internal node and has at least one leaf as a son.

Definition 2.2 Let f' be the \$-free-tree corresponding to an f-tree f. The three well-formedness conditions are defined as follows :

1. An f-tree f is said to be *unique* if, f' consistently represents a function.
2. An f-tree f is said to be *complete* if, for an arbitrary node n in f' labeled by $PRED$ ($PRED \in FN$), the following condition holds : when the unique son of n is labeled by $p(A_1, \ldots, A_k)$, each $A_i (1 \leq i \leq k)$ is equal to a label of some n's brother which is not neither a pre-terminal node nor a leaf.
3. An f-tree f is said to be *coherent* if, for an arbitrary node n in f' labeled by $PRED$ ($PRED \in FN$), the following condition holds : when the unique son of n is labeled by $p(A_1, \ldots, A_k)$, each label of an n's brother which is not neither a pre-terminal node nor a leaf is equal to A_i for some i, $1 \leq i \leq k$.

A terminal string x is *grammatical* only if it has a valid annotated phrase structure tree 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 needed to check the well-formedness of f-trees for all annotated phrase structure trees of x in general. In this case, if x is assigned at least one well-formed f-tree $f(x)$, then $x \in L(G)$.

The following theorem is known concerning the generative power of FRLFGs.

Theorem 2.1 [7] $CFL \subsetneq \mathcal{L}_{FRLFG} \subseteq CSL$ □

Here, CFL denotes the class of context-free languages, CSL denotes the class of context-sensitive languages, and $\mathcal{L}_{FRLFG}$ denotes the class of languages generated by FRLFGs. The following theorem is known concerning the complexity of the parsing problem for the FRLFG languages.

Theorem 2.2 [7] *Let G be an FRLFG, x be a terminal string with $|x| = n$, and Gr be the underlying CFG of G. Here, $|x|$ denotes the length of the string x.*

1. *If Gr is ambiguous, there is an algorithm deciding whether $x \in L(G)$ and if so, generating all different annotated phrase structure trees and f-trees for x in $O(n^3 + d(x, Gr) \cdot n^2)$ time, where $d(x, Gr)$ denotes the number of the different derivation trees for x, called degree of ambiguity.*
2. *If Gr is unambiguous, there is an algorithm deciding whether $x \in L(G)$ and if so, generating the unique annotated phrase structure tree and f-tree for x in $O(n^2)$ time.* □

It is known that, for any x and Gr, $d(x, Gr)$ is decidable using formal power series [9]. Note that $d(x, Gr)$ may be a function of $|x|$.

3 A Parsing Algorithm for the FRLFG Languages

In this section, we show an efficient parsing algorithm for the FRLFG languages, which consists of the next three algorithms : (1) an extended Earley's algorithm; (2) an f-tree construction algorithm; and (3) a well-formedness checking algorithm. The algorithm (1) is used to parse context-free languages. Then, the algorithm (2) constructs f-trees using the results of the algorithm (1) and functional assignments attached to context-free rules. Finally, the algorithm (3) tests whether the f-trees constructed by the algorithm (2) are well-formed. In Fig. 2, we show our parsing algorithm for the FRLFG languages.

3.1 An Extended Earley's Algorithm

It is well known that the Earley's algorithm is an efficient parsing algorithm for the context-free languages [1, 2, 4]. Since the Earley's algorithm is not designed to generate all derivation trees for an input string, we modified the Earley's algorithm to generate all derivation trees for every input string.

It can be shown that, for any string w, the number of derivations for w is finite if G is cycle-free, even when G includes ε-rules. Namely, we can prove the following theorem.

Theorem 3.1 *Let G be a CFG including ε-rules. If G is cycle-free, for an arbitrary string w, the number of derivations for w in G is finite.* □

Let Γ be a set of production names or an empty string ε. An extended Earley's algorithm constructs *extended parse lists* $I_0, \ldots, I_n$ in this order, where $n \geq 1$ is the length of an input string. For each j $(0 \leq j \leq n)$, an element of I_j is of the following form:

$$([A \rightarrow \alpha \cdot \beta, i], D),$$

where, $A \in NA$, $\alpha, \beta \in (NA \cup TA)^*$, $D \subseteq \Gamma^*$, and $0 \leq i \leq j$. This algorithm records necessary information in extended parse lists, in order to generate all derivation trees for an input string if any.

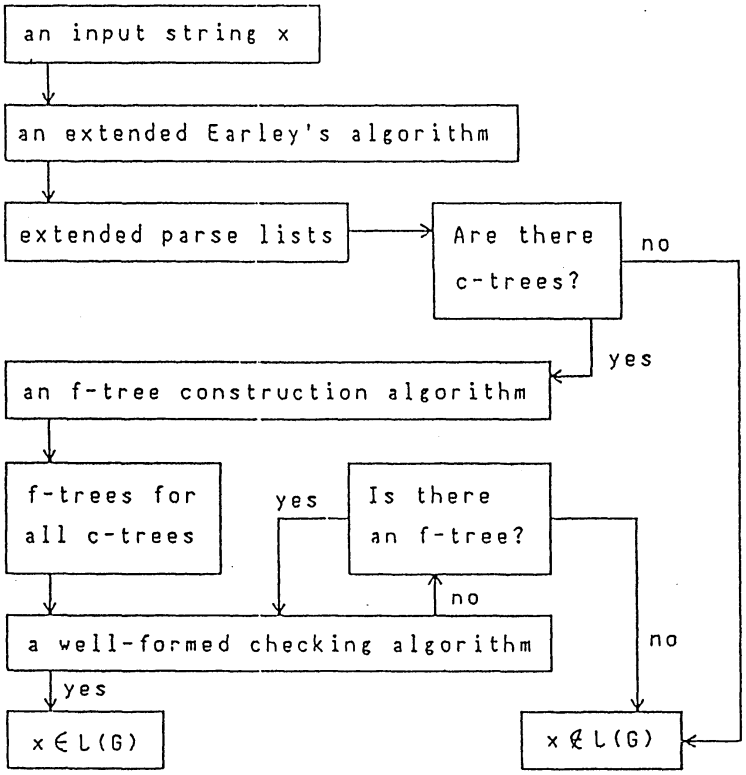

Figure 2: A parsing algorithm for FRLFG languages.

3.2 An F-tree Construction Algorithm

Let us assume that an element $([S \rightarrow \alpha \cdot, 0], D)$ is recorded in the n-th extended parse list by the extended Earley's algorithm, where

n is the length of an input string ($n \geq 1$),

S is a start symbol,

$\alpha \in (T\Lambda \cup N\Lambda)^*$,

$S \rightarrow \alpha \in P$, and

$D \subseteq \Gamma^+$.

Using elements of D and functional assignments, this algorithm evaluates functional assignments at each node by traversing an annotated phrase structure tree in *preorder* (top-down left-to-right order), and constructs f-trees for all c-trees.

4 Conclusion

In this paper, we have shown an efficient parsing algorithm for FRLFG languages. We have also implemented this algorithm by using C programming language. In order to parse sentences efficiently, it is necessary to decrease the degree of ambiguity as much as possible.

References

[1] Aho, A. V., and Ullman, J. D., *The Theory of Parsing, Translation, and Compiling*, Prentice-Hall Inc., Englewood Cliffs (1972).

[2] Aho, A. V., Sethi, R., and Ullman, J. D., *Compilers*, Addison-Wesley Publishing (1986).

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

[4] Moll, R. N., Arbib, M. A., and Kfoury, A. J., *An Introduction to Formal Language Theory*, Springer-Verlag (1988).

[5] 中西隆一, 関浩之, 嵩忠雄,「語彙機能文法の生成能力について」, '91 夏の LA シンポジウム資料 (1991).

[6] Nishino, T., An efficiently Parsable and Learnable Subclass of Lexical Functional Grammars, *IEICE Technical Report*, 90:25, pp.55-64 (1990).

[7] Nishino, T., *Formal Methods in Natural Languages Syntax*, Doctoral Dissertation, Waseda University (1991).

[8] Nishino, T., Shimizu, N., Yamada, S., and Yaku, T., On Normal Forms and Decision Problem for Lexical-Functional Grammars, *IEICE Technical Report*, 90:93, pp.21-32 (1991).

[9] Révész, *Introduction to Formal Languages*, McGraw-Hill Book Company(1983).