数理解析研究所講究録 1426 巻 2005 年 32-38

$O(n^3)$ で認識される文脈自由木言語のサブクラスについて

電気通信大学大学院 電気通信学研究科 川原田 郁雄 (Ikuo Kawaharada)
Graduate School of Electro-Communications, University of Electro-Communications
ikuo@calvyn.cs.uec.ac.jp

茨城大学 工学部 情報工学科 藤芳 明生 (Akio Fujiyoshi)
Department of Computer and Information Sciences, Ibaraki University
fujiyoshi@cis.ibaraki.ac.jp

Abstract

In this paper, the recognition algorithm for the class of tree languages generated by linear, monadic context-free tree grammars (LM-CFTGs) is proposed. LM-CFTGs define an important class of tree languages because LM-CFTGs are closely related to tree adjoining grammars (TAGs). The algorithm uses the CKY algorithm as a subprogram and recognizes whether an input tree can be derived from a given LM-CFTG in $O(n^3)$ time. With this fast recognition algoritm, we think that LM-CFTGs and spine grammars (an equivalent formalism) are useful in many applications.

1 Introduction

Tree structures are used extensively in computer science to represent hierarchical organizations. For the purpose of formalizing and classifying tree structures, many kinds of formalisms for tree structures have been proposed. One of the most popular formalisms for tree structures is regular tree grammars (RTGs)[1]. RTGs are a generalization of regular grammars (RGs) from strings to rooted, ordered, labeled trees. The class of tree languages generated by RTGs coincides with that accepted by deterministic bottom-up tree automata, and so trees generated by a RTG are recognized in $O(n)$ time, where n is the number of nodes of a tree. The class of tree languages generated by RTG is closely related to many applications, e.g., XML document validation and recognition of derivation trees of context-free grammars. However, for other applications such as representation of RNA secondary structure and formalisms for natural languages, a larger class of tree structures which includes more complicated sorts of trees needs to be considered. The generalization of context-free grammars (CFGs) from strings to rooted, ordered, labeled trees is context-free tree grammars (CFTGs)[7]. Though CFTGs formalize a larger class of tree structures than RTGs, it is thought that they are too large for application because the string languages generated by CFTGs are exactly indexed languages, whose emptiness problem and uniform membership problem are exponential time complete. However, there exists a restricted version of CFTGs which generates an interesting class of tree languages.

In this paper, the class of tree languages generated by linear, monadic CFTGs (LM-CFTGs)[2, 4] is considered, and the $O(n^3)$-time recognition algorithm for trees generated by a LM-CFTG is proposed. LM-CFTGs are CFTGs with nonterminals of rank 0 and 1 only and with at most one occurrence of a variable in every right-hand side of a production for a nonterminal of rank 1. It is known that LM-CFTGs generate the same class of string languages as tree adjoining grammars (TAGs)[5, 6, ?] and a strictly larger class of trees than TAGs. TAGs are another formalism for tree structures which have been widely studied relating them to natural languages. It is also noteworthy that LM-CFTGs are equivalent to spine grammars (SGs)[2, 3], which are also a restricted version of CFTGs but the restrictions are looser.

The recognition algorithm presented in this paper uses the CKY algorithm as a subprogram and recognizes whether an input tree can be derived from a given LM-CFTG in $O(n^3)$ time. For the purpose of easier understanding, an algorithm which recognizes marked trees will be presented first. A marked tree is a tree with marks on some nodes which are used to trace the derivation of the tree by the algorithm. Because the CKY algorithm can be directly applied to a marked tree, it can be easily confirmed whether the algorithm works correctly and its run time is $O(n^3)$ in the worst case. Then, the algorithm will be extended so that it will try to guess which nodes should have been marked and recognizes an input tree without marks. It is shown that the extended algorithm works correctly and its run time is also $O(n^3)$.

2 Preliminaries

In this section, terms, definitions, and former results which will be used in the rest of this paper are introduced.

Let $\mathcal{N}$ be the set of all natural numbers, and let $\mathcal{N}_+$ be the set of all positive integers. The concatenation operator is denoted by '$\cdot$'. For an alphabet Σ, the set of strings over Σ is denoted by Σ^*, and the empty string is denoted by λ.

2.1 Ranked Alphabets, Trees, Substitution and Context-Free Grammars

A *ranked alphabet* is a finite set of symbols in which each symbol is associated with a natural number, called the *rank* of a symbol. Let Σ be a ranked alphabet. For $a \in \Sigma$, the rank of a is denoted by $rank(a)$. For $n \geq 0$, it is defined that $\Sigma_n = \{a \in \Sigma \mid rank(a) = n\}$.

A set D is a *tree domain* if D is a nonempty finite subset of $(\mathcal{N}_+)^*$ satisfying the following conditions:

- For any $d \in D$, if $d', d'' \in (\mathcal{N}_+)^*$ and $d = d' \cdot d''$, then $d' \in D$.
- For any $d \in D$ and $i, j \in \mathcal{N}_+$, if $i \leq j$ and $d \cdot j \in D$, then $d \cdot i \in D$.

Let D be a tree domain and $d \in D$. Elements in D are called *nodes*. A node d' is a *child of d* if there exists $i \in \mathcal{N}_+$ such that $d' = d \cdot i$. A node is called a *leaf* if it has no child. The node λ is called the *root*. A node that is neither a leaf nor the root is called an *internal node*. A *path* is a sequence of nodes with the separator # such that $d_0 \# d_1 \# \cdots \# d_n, n \geq 0$ where $d_0, d_1, \ldots, d_n \in D$ and for $0 \leq i \leq n-1$, d_{i+1} is a child of d_i. Let Σ be a ranked alphabet. A *tree* over Σ is a function $\alpha : D \to \Sigma$ where D is a tree domain. The set of trees over Σ is denoted by T_Σ. The domain of a tree α is denoted by D_α. For $d \in D_\alpha$, $\alpha(d)$ is called the *label* of d. The *subtree* of α *at d* is $\alpha/d = \{(d', a) \in (\mathcal{N}_+)^* \times \Sigma \mid (d \cdot d', a) \in \alpha\}$.

The expression of a tree over Σ is defined to be a string over elements of Σ, parentheses and commas. For $\alpha \in T_\Sigma$, if $\alpha(\lambda) = b$, $\max\{i \in \mathcal{N}_+ \mid i \in D_\alpha\} = n$ and for each $1 \leq i \leq n$, the expression of α/i is α_i, then the expression of α is $b(\alpha_1, \alpha_2, \ldots, \alpha_n)$. Note that n is the number of the children of the root. For $b \in \Sigma_0$, trees are written as b instead of $b()$.When the expression of α is $b(\alpha_1, \alpha_2, \ldots, \alpha_n)$, it is written that $\alpha = b(\alpha_1, \alpha_2, \ldots, \alpha_n)$, i.e., each tree is identified with its expression.

Let Σ be a ranked alphabet, and let I be a set that is disjoint from Σ. $T_\Sigma(I)$ is defined to be $T_{\Sigma \cup I}$ where $\Sigma \cup I$ is the ranked alphabet obtained from Σ by adding all elements in I as symbols of rank 0. Let $X = \{x_1, x_2, \ldots\}$ be the fixed countable set of variables. It is defined that $X_0 = \emptyset$ and for $n \geq 1$, $X_n = \{x_1, x_2, \ldots, x_n\}$. x_1 is situationally denoted by x. Let $\alpha, \beta \in T_\Sigma$ and $d \in D_\alpha$. It is defined that $\alpha\langle d \leftarrow \beta\rangle = \{(d', a) \mid (d', a) \in \alpha$ and d is not a prefix of $d'\} \cup \{(d \cdot d'', b) \mid (d'', b) \in \beta\}$, i.e., the tree $\alpha\langle d \leftarrow \beta\rangle$ is the result of replacing α/d by β. Let $\alpha \in T_\Sigma(X_n)$, and let $\beta_1, \beta_2, \ldots, \beta_n \in T_\Sigma(X)$. The notion of substitution is defined. The result of substituting each β_i for nodes labeled by variable x_i in α, denoted by $\alpha[\beta_1, \beta_2, \ldots, \beta_n]$, is defined as follows.

- If $\alpha = a \in \Sigma_0$, then $a[\beta_1, \beta_2, \ldots, \beta_n] = a$.
- If $\alpha = x_i \in X_n$, then $x_i[\beta_1, \beta_2, \ldots, \beta_n] = \beta_i$.
- If $\alpha = b(\alpha_1, \alpha_2, \ldots, \alpha_k)$, $b \in \Sigma_k$ and $k \geq 1$, then
  $\alpha[\beta_1, \beta_2, \ldots, \beta_n] = b(\alpha_1[\beta_1, \beta_2, \ldots, \beta_n], \ \ldots \ , \alpha_k[\beta_1, \beta_2, \ldots, \beta_n])$.

A *context-free grammar* (CFG) is a 4-tuple $\mathcal{G} = (N, T, P, S)$, where: N and T are disjoint alphabets of *nonterminals* and *terminals*, respectively, P is a finite set of *productions* of the form $A \to \gamma$ where $\gamma \in (N \cup T)^*$, and $S \in N$ is the *initial nonterminal*. The *language generated by* $\mathcal{G}$, denoted by $L(\mathcal{G})$, is defined as usual.

2.2 Context-Free Tree Grammars

The context-free tree grammars (CFTGs) were introduced by W. C. Rounds [7] as tree generating systems. The definition of CFTGs is a direct generalization of context-free grammars.

Definition 1 A *context-free tree grammar* (CFTG) is a 4-tuple $\mathcal{G} = (N, \Sigma, P, S)$, where: N and Σ are disjoint ranked alphabets of *nonterminals* and *terminals*, respectively, P is a finite set of *productions* of the form $A(x_1, x_2, \ldots, x_n) \rightarrow \alpha$, with $n \geq 0$, $A \in N_n$, $\alpha \in T_{N \cup \Sigma}(X_n)$, and $S \in N_0$ is the *initial nonterminal.*

Definition 2 For a CFTG $\mathcal{G}$, $\underset{\mathcal{G}}{\Rightarrow}$ is the relation on $T_{\Sigma \cup N} \times T_{\Sigma \cup N}$ such that for a tree $\alpha \in T_{\Sigma \cup N}$ and a node $d \in D_\alpha$, if $\alpha/d = A(\alpha_1, \alpha_2, \ldots, \alpha_n)$ where $A \in N_n$, $\alpha_1, \ldots, \alpha_n \in T_{\Sigma \cup N}$, and $A(x_1, x_2, \ldots, x_n) \rightarrow \beta$ is in P, then $\alpha \underset{\mathcal{G}}{\Rightarrow} \alpha\langle d \leftarrow \beta[\,\alpha_1, \alpha_2, \ldots, \alpha_n]\rangle$. A *derivation* is a finite sequence of trees $\alpha_0 \alpha_1 \cdots \alpha_n$ such that $n \geq 0$ and $\alpha_0 \underset{\mathcal{G}}{\Rightarrow} \alpha_1 \underset{\mathcal{G}}{\Rightarrow} \cdots \underset{\mathcal{G}}{\Rightarrow} \alpha_n$. When there exists a derivation $\alpha_0 \alpha_1 \cdots \alpha_n$, we write $\alpha_0 \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha_n$. The subscript $\mathcal{G}$ on the relations $\underset{\mathcal{G}}{\Rightarrow}$ and $\overset{*}{\underset{\mathcal{G}}{\Rightarrow}}$ may be dropped when $\mathcal{G}$ is clearly understood. The *tree language generated by* $\mathcal{G}$ is the set $L(\mathcal{G}) = \{\alpha \in T_\Sigma \mid S \overset{*}{\underset{\mathcal{G}}{\Rightarrow}} \alpha\}$. The *string language generated by* $\mathcal{G}$ is $L_S(\mathcal{G}) = \{\mathit{yield}(\alpha) \mid \alpha \in L(G)\}$. $\mathcal{G}_1$ and $\mathcal{G}_2$ are *equivalent* if $L(\mathcal{G}_1) = L(\mathcal{G}_2)$. $\mathcal{G}_1$ and $\mathcal{G}_2$ are *weakly equivalent* if $L_S(\mathcal{G}_1) = L_S(\mathcal{G}_2)$.

Linear, monadic CFTGs (LM-CFTGs) are CFTGs with two restrictions. One is 'linear' that requires the number of occurrences of every variable in the right-hand side of a production be no more than 1, and the other is 'monadic' that requires the rank of nonterminals be either 0 or 1.

Definition 3 A CFTG $\mathcal{G} = (N, \Sigma, P, S)$ is *monadic* if the rank of any nonterminal is either 0 or 1, i.e., $N = N_0 \cup N_1$ and $N_n = \emptyset$ for $n \geq 2$. $\mathcal{G}$ is *linear* if for any production $A(x_1, x_2, \ldots, x_n) \rightarrow \alpha$ in P, no variable occurs more than once in α.

3 Properties of LM-CFTGs

In this section, we discuss formal properties of LM-CFTGs that give us intuition about the class of trees generated by LM-CFTGs.

First, we introduce normal form of LM-CFTGs. The notion of normal form of LM-CFTGs is analogous to that of Chomsky normal form of CFGs. For any LM-CFTG, there exists an equivalent LM-CFTG $\mathcal{G} = (N, \Sigma, P, S)$ that satisfies the following conditions:

1. For each $A \in N_0$, if $A \rightarrow \alpha$ is in P, then either $\alpha = a$ with $a \in \Sigma_0$, or $\alpha = B(C)$ with $B \in N_1$ and $C \in N_0$.
2. For each $A \in N_1$, if $A(x) \rightarrow \alpha$ is in P, then either $\alpha = B(C(x))$ with $B, C \in N_1$ or $\alpha = b(C_1, \ldots, C_{i-1}, x, C_{i+1}, \ldots, C_m)$ with $n \geq 0$, $b \in \Sigma_{n+1}$ and $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_m \in N_0$.

If a LM-CFTG satisfies the above conditions, it is said that the grammar is in *normal form.* The definition of normal form is slightly different from the one introduced in [2] but it is easy to obtain this normal form in the same way as the construction of the Chomsky normal form of context-free grammars.

For an LM-CFTG in normal form, productions which grow a tree horizontally are only of the form $A(x) \rightarrow b(C_1, \ldots, C_{i-1}, x, C_{i+1}, \ldots, C_n)$. In the right-hand side of the rules, the i-th child of b is different from the other children. Thus, we introduce modified LM-CFTGs called marking LM-CFTGs whose productions are with a marker that can be used to trace derivations.

Definition 4 For $\mathcal{G} = (N, \Sigma, P, S)$ be an LM-CFTG in normal form, the *marking LM-CFTG* associated with $\mathcal{G}$, $\mathcal{G}' = (N \cup \overline{N}, \Sigma \cup \overline{\Sigma}, P', S)$, is defined as follows. $\overline{N} = \{\overline{A} \mid A \in N\}$, $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$, and P' is the smallest set of productions safisfing the following conditions:

1. If $A \rightarrow a$ is in P for some $A \in N_0$ and $a \in \Sigma_0$, then $A \rightarrow a$ and $\overline{A} \rightarrow \overline{a}$ are in P'.
2. If $A \rightarrow B(C)$ is in P for some $A, C \in N_0$ and $B \in N_1$, then $\overline{A} \rightarrow \overline{B}(\overline{C})$ and $A \rightarrow B(\overline{C})$ are in P'.
3. If $A(x) \rightarrow B(C(x))$ is in P for some $A, B, C \in N_1$, then $\overline{A}(x) \rightarrow \overline{B}(\overline{C}(x))$ and $A(x) \rightarrow B(\overline{C}(x))$ are in P'.
4. If $A(x) \rightarrow b(C_1, \ldots, C_{i-1}, x, C_{i+1}, \ldots, C_m)$ is in P for some $m \geq 0, b \in \Sigma_m$ and $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_m \in N_1$, then $\overline{A}(x) \rightarrow \overline{b}(C_1, \ldots, C_{i-1}, x, C_{i+1}, \ldots, C_m)$ and $A(x) \rightarrow b(C_1, \ldots, C_{i-1}, x, C_{i+1}, \ldots, C_m)$ are in P'.

Elements in $\overline{N} \cup \overline{\Sigma}$ are called *marked,* and elements in $N \cup \Sigma$ are called *un-marked.*

Definition 5 Let $\mathcal{G}$ be a marking LM-CFTG. For a tree $\alpha \in L(\mathcal{G})$ and a node $d_0 \in D_\alpha$, the *marked path* starts from d_0 is a path $d_0\#d_1\#\cdots\#d_n$ such that $n \geq 0$, d_n is a leaf, and for $1 \leq i \leq n$, $\alpha(d_i)$ is marked. When d_0 is the root of α, the marked path starts from d_0 is called the *spine* of α. When $\alpha(d_0)$ is un-marked, the marked path starts from d_0 is called a *subspine* of α.

From the above definitions, the following claim is immediate.

Claim 1 Let $\mathcal{G}$ be a marking LM-CFTG. For any tree derived by $\overline{\mathcal{G}}$,

- each node has exactly one marked child if the node is not a leaf,
- an un-marked node is either the root or a child of a marked node, and
- there exists exactly one marked path for each node.

Definition 6 For an LM-CFTG $\mathcal{G} = (N, \Sigma, P, S)$, the spine producing CFG associated with $\mathcal{G}$, $\overline{\mathcal{G}} = (N, \Sigma, P', S)$, is defined as follows. P' is the smallest set of productions satisfying the following conditions:

1. If a production $A \to a$ is in P for some $A \in N_0, a \in \Sigma_0$, then $A \to a$ is in P',
2. If a production $A \to B(C)$ is in P for some $A, C \in N_0, B \in N_1$, then $A \to BC$ is in P',
3. If a production $A(x) \to B(C(x))$ is in P for some $A, B, C \in N_1$, then $A \to BC$ is in P', and
4. If a production $A(x) \to b(C_1, \ldots, C_{i-1}, x, C_{i+1}, \ldots, C_m)$ is in P for some $m \geq 0$, $A \in N_1, b \in \Sigma_m$ and $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_m \in N_0$, then $A \to b$ is in P'.

Lemma 1 Let $\mathcal{G} = (N \cup \overline{N}, \Sigma \cup \overline{\Sigma}, P, S)$ be a marking LM-CFTG, and let $\overline{\mathcal{G}} = (N \cup \overline{N}, \Sigma \cup \overline{\Sigma}, P', S)$ be the spine producing CFG associated with $\mathcal{G}$. For any tree $\alpha \in L(\mathcal{G})$, $d_0\#d_1\#\cdots\#d_n$ is a subspine of α if and only if $\alpha(d_0) \cdot \alpha(d_1) \cdot \cdots \cdot \alpha(d_n) \in L(\overline{\mathcal{G}})$.

Proof. From the construction of $\overline{\mathcal{G}}$, the lemma clearly holds. □

4 Recognition Algorithms

4.1 Parsing Algorithms for Marking LM-CFTGs

We assume that a marking LM-CFTG $\mathcal{G} = (N \cup \overline{N}, \Sigma \cup \overline{\Sigma}, P, S)$ is given and the CFG $\overline{\mathcal{G}} = (N \cup \overline{N}, \Sigma \cup \overline{\Sigma}, P', S)$ is constructed from $\mathcal{G}$. The algorithm consists of three parts: Parse-Tree, Parse and CKY. First, the function Parse-Tree takes an input and initializes variables. Next, the main procedure Parse is invoked. The procedure Parse is defined recursively and checks a tree in bottom-up by using the function CKY. For a given tree, the procedure Parse stores a marked path. The function CKY is based on the CKY algorithm for context-free languages. The function CKY checks whether a subspine stored by the function Parse could be generated by the CFG $\overline{\mathcal{G}}$.

The Function Parse-Tree

The function Parse-Tree takes as input a tree α. This function prepare a variable τ_d and a set U_d for each node d in α. The variable τ_d stores a marked path of the node d. This string shows the path with the marker from some leaf to the node d. In the 5th line of this function, $CKY(\tau_\lambda, S)$ is invoked. $CKY(\tau_\lambda, S)$ checks whether a spine of the root node τ_λ could be derived from initial symbol S by the CFG $\overline{\mathcal{G}}$.

```
Parse-Tree:
Input a tree $\alpha$. Output accept if $\alpha \in L(\mathcal{G})$ otherwise reject.
begin
1  For each node $d$ in $D_\alpha$, we prepare the set $U_d$ and
   the variable $\tau_d$ over strings of nodes with the concatenation operator $\#$ .
2  Initialize the variable $\tau_d := \lambda$ and the set $U_d := \emptyset$ for each node $d$ in $D_\alpha$.
3  For each node $d$ in $\alpha$, check that $d$ has exactly one marked child if $d$ is not a leaf,
   otherwise reject.
4  $Parse(\alpha, \lambda)$
5  if $CKY(\tau_\lambda, S) = true$, then return accept else return reject.
end
```

The Procedure Parse

The procedure Parse takes as input a tree and a node. The given node shows where the given tree is located in the whole tree, because this procedure is defined recursively. The algorithm works by parsing in a bottom-up way. This procedure stores a marked path of the given node. And nonterminals of the left side symbol of productions which derive the root node and un-marked children are stored.

```
Parse:
Input a tree $\alpha$ and a node $d$.
begin
1  Let $\alpha(\lambda) = b$ and $rank(b) = m$.
2  if $m = 0$, then
3      $\tau_d := d$ and $U_d := \{A \mid A \to \alpha \in P\}$
   else begin
4      for $i := 1$ to $m$ do $Parse(\alpha/i, d{\cdot}i)$
5      Let the $h$-th child of the root be marked.
6      for each $A(x) \to b(C_1, \ldots, C_{h-1}, x, C_{h+1}, \ldots, C_m)$ is in $P$ for some $A \in N_1 \cup \overline{N}_1$,
           $C_1, \ldots, C_{h-1}, C_{h+1}, \ldots, C_m \in N_0$,
           if $CKY(\tau_{d \cdot j}, C_j) = true$, for all $j \in \{1, \ldots, h-1, h+1, \ldots, m\}$,
       then $U_d := U_d \cup \{A\}$
7      $\tau_d := d\#\tau_{d \cdot h}$
8  end
end
```

The Function CKY

The extended CKY takes as an input a path which is a subspine of some node, and check whether the given path could be derived by the CFG $\overline{\mathcal{G}}$. This algorithm is the same way as well-known recognition algorithm for context-free languages.

```
CKY:
Input a path $\tau$ and a nonterminal $A$. Output $true$ or $false$.
begin
1  The path $\tau$ can be written as $d_1\#d_2\#\cdots\#d_m$ where $d_1, d_2, \ldots, d_m \in D_\alpha$ and $m \geq 1$.
2  Let $V$ be an $m \times m$ matrix.
3  for $i := 1$ to $m$ do
4      $V_{i,1} := U_{d_i}$
5  for $j := 2$ to $m$ do
6      for $i := 1$ to $m - j + 1$ do begin
7          $V_{i,j} := \emptyset$
8          for $k := 1$ to $j - 1$ do
9              $V_{i,j} := V_{i,j} \cup \{A \mid A \to BC \in P', B \in V_{i,k}, C \in V_{i+k,j-k}\}$
       end
10 if $A \in V_{1,m}$, then return $true$ else return $false$
end
```

From above algorithms, we obtain the following results. We leave these proofs of theorems to the reader .

Theorem 1 $A \overset{*}{\Rightarrow} \alpha \in T_\Sigma$ if and only if $CKY(\tau_\lambda, A) = true$ after applying $Parse(\alpha, \lambda)$.

Theorem 2 Our recognition algorithm runs in $O(n^3)$ time where n is the number of nodes of the input tree.

4.2 Parsing Algorithms for general LM-CFTGs

In this section, we present algorithms for tree languages generated by general LM-CFTGs. In above section, LM-CFTGs are marked. So, we can distinguish paths which generated by using productions of the form $A \to B(C)$ and $A(x) \to B(C(x))$, and check these paths in the CKY style algorithm. The new algorithm should guess these marked paths.

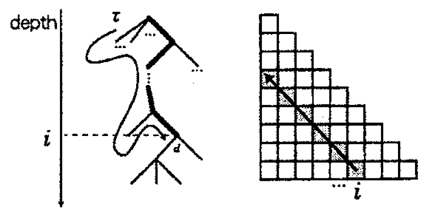

Figure 1: The CKY algorithm for parsing the tree

Since there is no danger of confusion, we use the same labels for functions and procedures. The function Parse-Tree takes an input and initializes variables. For each node d in the input tree, the function prepare a set U_d. The set U_d will be set by the procedure Parser. At the node d, if the algorithm guess i-th child to be marked, then nonterminals will be added to the set $U_{d\cdot i}$. So, this function prepare another sets for nodes of leaves.

```
Parse-Tree:
Input a tree $\alpha$. Output accept if $\alpha \in L(\mathcal{G})$ otherwise reject.
begin
1   For all node $d$ in $D_\alpha$, we prepare the set $U_d := \emptyset$ and the tree $\tau_d := \lambda$.
2   For each leaf $d$ in $D_\alpha$, we prepare the set $U'_d := \emptyset$.
3   $Parse(\alpha, \lambda)$
4   if $CKY(\tau_\lambda, S) = true$, then return accept else return reject.
end
```

The procedure Parse behaves in the same way of above marked version, but nonterminals is added to the set $U_{d\cdot i}$, if this procedure guess i-th child to be marked. If the input tree has no child, then nonterminals which derived this node are stored in the set U'_d. This extension algorithm needs at most constant times of above version.

```
Parse:
Input a tree $\alpha$ and a node $d$.
begin
1   Let $\alpha(\lambda) = b$ and $rank(b) = m$.
2   if $m = 0$, then
3       $U'_d := \{A \mid A \to \alpha \in P\}$
    else begin
4       for $i := 1$ to $m$ do $Parse(\alpha/i, d\cdot i)$
5       for each $A(x) \to b(C_1, \ldots, C_{i-1}, x, C_{i+1}, \ldots, C_m)$ is in $P$ for some $A \in N_1$,
            $C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_m \in N_0$, and $1 \leq i \leq m$,
            if $CKY(\alpha/j, C_j) = true$, for all $j \in \{1, \ldots, i-1, i+1, \ldots, m\}$
            then $U_{d\cdot i} := U_{d\cdot i} \cup \{A\}$
    end
end
```

We extend CKY to parsing the tree structure. The function CKY takes as input a tree τ and checks whether there exists a path from the root to a leaf derivable by CFG $\overline{\mathcal{G}}$. The following algorithm search the input tree in the depth-first search and construct a matrix V. This algorithm sets result to set W whenever reaching a leaf of the tree. And last, CKY checks whether the nonteminal A is in the set W. This algorithm reuse cells corresponded to prefix path and for each node d, sets cells to upper left neighbor (see Figure 1). Let n be a number of nodes of the input tree. For each node, this algorithm sets sells at most n times. Each cell needs to set in $O(n)$ time. This algorithm constructs cells exactly once for same node, because this algorithm visits nodes in depth-first search. So, the total time complexity is $O(n^3)$.

```
CKY:
Input a tree $\tau$ and a nonterminal $A$. Output true or false.
begin
1   Let $m$ be the number of nodes in $\tau$.
```

```
     Prepare the $m \times m$ matrix $V$ and the set $W$.
2    Initialize $W := \emptyset$
3    This algorithm visits each node in the tree $\tau$ in the depth-first search
     and sets the matrix $V$ in the following way:
4    for each node $d$ (whose depth is $i = |d| + 1$)
     begin
5        if $d$ has no child then, $V_{i,1} := U'_d$
         else
6            Let the next visiting node be $\ell$-th child of $d$.
7            $V_{i,1} := U_{d \cdot \ell}$
8        for $j := 1$ to $i - 1$ do begin
9            $V_{i-j,j+1} := \emptyset$
10           for $k := 1$ to $j$ do
11               $V_{i-j,j+1} := V_{i-j,j+1} \cup \{A \mid A \to BC \in P', B \in V_{i-j,j-k+1}, C \in V_{j-k+1,k}\}$
         end
12       if $d$ is a leaf, then $W := W \cup V_{1,i}$
     end
13   if $A \in W$, then return *true* else return *false*
end
```

The complexity does not increase by extension of algorithms. Thus, we have the following result.

Theorem 3 A recognition of tree languages generated by LM-CFTGs is in time $O(n^3)$, where n is the number of nodes of the input tree.

5 Conclusion

In this paper, we present an $O(n^3)$ time algorithm for recognition tree languages generated by LM-CFTGs. This algorithm arise from properties of LM-CFTGs such that a set of subspines and the other productions rewrite a nonterminal on the paths is a context-free lanuage. Thus, we can apply a CKY-like algorithm to these paths. Our recognition algorithm might be a little slow in an actual application. However, we conjecture that a limitation could be introduced into LM-CFTGs as in the case of LR-grammars to construct more faster recognition algorithms, i.e., productions which derive subspines are limited to generate a 'deterministic' context-free lanuage. In this paper, we have not be able to present recognition algorithms for linear CFTGs. The class of tree sets generated by linear CFTGs is closely related to that of multicomponent TAGs. How order time is necessary for recognition of tree languages generated by linear CFTGs? We believe that a solution for these questions will make the notion of the tree language recognitions more interesting.

References

[1] W. S. Brainerd, Tree generating regular systems, Information & Control 14 (2) (1969) 217–231.

[2] A. Fujiyoshi, T. Kasai, Spinal-formed context-free tree grammars, Theory of Computing Systems 33 (1), 2000, pp. 59–83.

[3] A. Fujiyoshi, Epsilon-free grammars and lexicalized grammars that generate the class of the mildly context-sensitive languages, in: TAG+7, Vancouver, 2004, pp. 16–23.

[4] A. Fujiyoshi, Linearity and nondeletion on monadic contex-free tree grammars, in: Information Processing Letters, 2004, pp. 103–107.

[5] A. K. Joshi, L. S. Levy, M. Takahashi, Tree adjunct grammars, J. Computer & System Sciences 10 (1), 1975, pp. 136–163.

[6] A. K. Joshi, Y. Schabes, Handbook of Formal Languages, Vol. 3, Springer, Berlin, 1996, Ch. Tree-adjoining grammars, pp. 69–124.

[7] W. C. Rounds, Mapping and grammars on trees, Mathematical Systems Theory 4 (3), 1970, pp. 257–287.