Multiple Context-Free Grammars

Daniel Norbert Götzmann

Saarland University

Abstract. Multiple context-free grammar (MCFG) is a weakly context-sensitive grammar formalism that deals with tuples of strings. An MCFG is called m-MCFG if all tuples have at most m components. In this summary I will outline that the expressivity of m-MCFG's increases with the parameter m and that the class of tree-adjoining languages is properly included in the class of 2-multiple context-free languages. In addition, I will give a brief overview over the parsing complexity of m-MCFG's.

1 Generalized context-free grammars

Multiple context-free grammars are a restricted form of generalized context-free grammars. Generalized context-free grammar is a grammar formalism that deals with tuples of strings. Rewriting rules in generalized context-free grammars have the form

$$A \to f[B^{(1)}, B^{(2)}, \ldots, B^{(q)}]$$

where A and $B^{(1)}, \ldots, B^{(q)}$ are nonterminal symbols and f is a partial function whose arguments and function values are tuples of strings. Formally, a generalized context-free grammar is defined as follows.

Definition 1. *A generalized context-free grammar (GCFG) is a 5-tuple* $\mathcal{G} = (N, O, F, P, S)$.

- N *is a finite set of nonterminal symbols.*
- O *is a set of n-tuples* $(n \geq 1)$ *over a finite set of symbols.*
- F *is a finite set of partial functions from* $O \times \cdots \times O$ *to* O.
- P *is a finite set of rewriting rules.*
- $S \in N$ *is the start symbol.*

The rewriting rules contained in P *are written as*

$$A \to f[B^{(1)}, B^{(2)}, \ldots, B^{(q)}]$$

where $A \in N$ *and* $B^{(1)}, B^{(2)}, \ldots, B^{(q)} \in N$ *are nonterminal symbols and* $f \in F$ *is a function from* O^q *to* O. *A rewriting rule is called terminating rule, if* $q = 0$ *holds, i.e. there is no terminal symbol in the right-hand side of this rewriting rule. In this case, the rewriting-rule is written as*

$$A \to \theta$$

where $\theta \in O$.

$A \to g_a[A]$ $g_a[(x_1)] = (x_1 a)$	$B \to g_b[B]$ $g_b[(x_1, x_2)] = (bx_1, bx_2)$	$S \to f[A, B]$ $f[(x_1), (y_1, y_2)] = (y_1 c^{\lvert x_1 \rvert^2} y_2)$
$A \to \theta_a$ $\theta_a = (\varepsilon)$	$B \to \theta_b$ $\theta_b = (\varepsilon, \varepsilon)$	
$L_\mathcal{G}(A)$ $= \{(a^m) \mid m \in \mathbb{N}\}$	$L_\mathcal{G}(B)$ $= \{(b^n, b^n) \mid n \in \mathbb{N}\}$	$L_\mathcal{G}(S)$ $= \{(b^n c^{m^2} b^n) \mid m, n \in \mathbb{N}\}$

Table 1. Rewriting rules, functions and derivations of Example 1

In context-free grammars, for each nonterminal there exists a set of words that can be derived from that nonterminal. Similarly, in generalized context-free grammars, for each nonterminal $A \in N$ there exists a set $L_\mathcal{G}(A)$ which contains exactly those tuples that can be derived from A in $\mathcal{G}$.

Definition 2. *For each nonterminal $A \in N$ the set $L_\mathcal{G}(A)$ is defined as the smallest set that satisfies the following two conditions:*

1. *For each terminating rule $A \to \theta \in P$, $\theta \in L_\mathcal{G}(A)$.*
2. *For each nonterminating rule $A \to f[B^{(1)}, B^{(2)}, \ldots, B^{(q)}] \in P$ and for each $\theta_i \in B^{(i)} (1 \leq i \leq q)$, if $f[\theta_1, \theta_2, \ldots, \theta_q]$ is defined, then $f[\theta_1, \theta_2, \ldots, \theta_q] \in L_\mathcal{G}(A)$.*

Every generalized context-free grammar $\mathcal{G}$ generates a language $L(\mathcal{G})$ which is the set that contains exactly those tuples that can be derived from the start symbol S in $\mathcal{G}$.

Definition 3. *Let $\mathcal{G} = (N, O, F, P, S)$ be a generalized context-free grammar. $L(\mathcal{G}) = L_\mathcal{G}(S)$ is called the generalized context-free language (GCFL) generated by the GCFG $\mathcal{G}$. It contains exactly those n-tuples that can be derived from the start symbol S in $\mathcal{G}$.*

Example 1. $\mathcal{G} = (N, O, F, P, S) = (\{S, A, B\}, (\{a, b, c\}^*)^+, \{f, g_a, g_b, \theta_a, \theta_b\}, \{S \to f[A, B], A \to g_a[A], A \to \theta_a, B \to g_b[B], B \to \theta_b\}, S)$ with

- $f(x, y) = \begin{cases} (y_1 c^{\lvert x_1 \rvert^2} y_2) & \text{if } y = (y_1, y_2) \text{ is a 2-tuple and } x = (x_1) \text{ is a 1-tuple} \\ \text{undefined} & \text{otherwise} \end{cases}$
- $g_a(x) = \begin{cases} (x_1 a) & \text{if } x = (x_1) \text{ is a 1-tuple} \\ \text{undefined} & \text{otherwise} \end{cases}$
- $g_b(x) = \begin{cases} (bx_1, bx_2) & \text{if } x = (x_1, x_2) \text{ is a 2-tuple} \\ \text{undefined} & \text{otherwise} \end{cases}$
- $\theta_a = (\varepsilon)$
- $\theta_b = (\varepsilon, \varepsilon)$

is a generalized context-free grammar that generates the generalized context-free language $L(\mathcal{G}) = \{b^n c^{m^2} b^n \mid m, n \in \mathbb{N}\}$.

Table 1 shows the connection between the rewriting rules, the functions and the resulting derivations. For the nonterminal A there are two rewriting rules in

P. $A \to \theta_a$ is a terminating rule, therefore $\theta_a = (\varepsilon)$ can be derived from A in $\mathcal{G}$. Since the rewriting rule $A \to g_a[A]$ is a nonterminating rule in N, if (x_1) can be derived from A in $\mathcal{G}$, the tuple $(x_1 a)$ can also be derived from A in $\mathcal{G}$. Intuitively, the tuple containing the empty word can be derived from A. By application of the nonterminating rule, arbitrarily many a's can be concatenated. Therefore, $L_{\mathcal{G}}(A) = \{a^m | m \in \mathbb{N}\}$.

For the nonterminal B there are also two rewriting rules in P. The terminating rule $B \to \theta_b$ implies $\theta_b = (\varepsilon, \varepsilon) \in L_{\mathcal{G}}(B)$. The nonterminating rule $B \to g_b[B]$ implies that if a 2-tuple $(x_1, x_2) \in L_{\mathcal{G}}(B)$ can be derived from B in $\mathcal{G}$, the 2-tuple (bx_1, bx_2) can also be derived from B in $\mathcal{G}$. The rules for B are similar to the rules for A except that the concatenation is parallel in two tuple components.

For the nonterminal S there is one rewriting rule in P, $S \to f[A, B]$. Therefore, for every 1-tuple $(x_1) \in L_{\mathcal{G}}(A)$ that can be derived from A and for every 2-tuple (y_1, y_2) that can be derived from B, the 1-tuple $(y_1 c^{|x_1|} y_2)$ can be derived from S in $\mathcal{G}$.

While the partial functions g_a and g_b in the previous example are rather simple functions which are defined as concatenation of components of their arguments and some constant strings, the partial function f is more complex.

According to Seki et al. [1], if a generalized context-free grammar may contain arbitrary partial recursive functions, then every language that can be generated by a Type 0 grammar can also be generated by a generalized context-free grammar. Therefore, generalized context-free grammars are very expressive.

However, restricting the functions that may be contained in generalized context-free grammars to functions that are defined as concatenation of constant strings and components of their arguments results in an interesting weakly context-sensitive subclass of GCFG's, the so-called multiple context-free grammars.

2 Multiple context-free grammars

Multiple context-free grammar is a subclass of generalized context-free grammar. Unlike generalized context-free grammar, it may not contain arbitrary partial functions but only partial functions that are defined as concatenation of constant strings and components of their arguments.

Definition 4. *An m-multiple context-free grammar (m-MCFG) is a generalized context-free grammar $\mathcal{G} = (N, O, F, P, S)$ that satisfies the following conditions:*

1. *$O = \bigcup_{i=1}^{m} (T^*)^i$, where T is a finite set of terminal symbols.*
2. *For each function $f \in F$, which takes $a(f)$ arguments, there are positive integers $r(f)$ and $d_1(f), \ldots d_{a(f)}(f)$ such that $1 \leq r(f) \leq m$ and $1 \leq d_i(f) \leq m$ $(1 \leq i \leq a(f))$, such that f is a function from $(T^*)^{d_1(f)} \times (T^*)^{d_2(f)} \times \cdots \times (T^*)^{(d_{a(f)}(f))}$ to $(T^*)^{r(f)}$*

3. *Each function $f \in F$ is defined as a concatenation of some constant strings in T^* and components of its arguments such that every component appears at most once in the result. More formally: For each function $f \in F$ there exist positive integers $v_h (1 \leq h \leq r(f))$, constant strings $\alpha_{(i,j)} \in T^* (1 \leq i \leq r(f), 0 \leq j \leq v_i)$ and an injective function ϕ_f from $\{(i,j) \in \mathbb{N} \times \mathbb{N} | 1 \leq i \leq r(f), 1 \leq j \leq v_i\}$ to $\{(i,j) \in \mathbb{N} \times \mathbb{N} | 1 \leq i \leq a(f), 1 \leq j \leq d_i(f)\}$, such that*

$$f[(x_{(1,1)}, \ldots, x_{1,d_1(f)}), \ldots, (x_{(a(f),1)}, \ldots, x_{a(f),d_{a(f)}(f)})] = (f_1, f_2, \ldots, f_{r(f)})$$

where every $f_h (1 \leq h \leq r(f))$ is of the form

$$f_h = \alpha_{(h,0)} x_{\phi_f(h,1)} \alpha_{(h,1)} x_{\phi_f(h,2)} \ldots x_{\phi_f(h,v_h(f))} \alpha_{(h,v_h(f))}$$

4. *For each nonterminal $A \in N$, there exists a positive integer $d(A)$, such that all tuples that can be derived from A in $\mathcal{G}$ have exactly $d(A)$ components.*
5. *$d(S) = 1$, i.e. all tuples that can be derived from the start symbol S are 1-tuples.*
6. *Every rewriting rule $A \to f[B^{(1)}, B^{(2)}, \ldots, B^{(q)}]$ must satisfy the condition $d(A) = r(f)$, i.e. the size of the tuple returned by f must be the same as the size of the tuples that can be derived from the nonterminal A. Additionally, it must satisfy the condition $d(B^{(i)}) = d_i(f) (1 \leq i \leq q)$, i.e. the size of the tuples that can be derived from $B^{(i)}$ must be the same as the size which f expects its i-th argument to have.*

The first condition restricts the tuples that are contained in an m-MCFG to tuples that have at most m elements. Section 3 deals with the influence of that parameter on the expressivity of m-multiple context-free grammars.

A language generated by an m-MCFG is called an m-multiple context-free language.

Definition 5. *Let $\mathcal{G} = (N, O, S, F, P)$ be an m-multiple context-free grammar. $L(\mathcal{G}) = L_{\mathcal{G}}(S)$ is called the m-multiple context-free language (m-MCFL) generated by the m-MCFG $\mathcal{G}$.*

Example 2. $\mathcal{G} = (N, O, F, P, S) = (\{S, A, B\}, (\bigcup_{i=1}^{2} \{a, b, c\}^*)^i, \{f_2, g_a, g_b, \theta_a, \theta_b\}, \{S \to f_2[A, B], A \to g_a[A], A \to \theta_a, B \to g_b[B], B \to \theta_b\}, S)$ with

- $f((x_1), (y_1, y_2)) = (y_1 x_1 y_2)$
- $g_a(x_1) = (x_1 a)$
- $g_b(x_1, x_2) = (b x_1, b x_2)$
- $\theta_a = (\varepsilon)$
- $\theta_b = (\varepsilon, \varepsilon)$

is a 2-multiple context-free grammar that generates the 2-multiple context-free language $L(\mathcal{G}) = \{b^n a^m b^n | m, n \in \mathbb{N}\}$.

Table 2 shows the connection between the rewriting rules, the functions and the resulting derivations. The rewriting rules for the nonterminals A and B are the same as in Example 1. They satisfy the conditions of Definition 4 because

$A \to g_a[A]$ $g_a[(x_1)] = (x_1 a)$	$B \to g_b[B]$ $g_b[(x_1, x_2)] = (bx_1, bx_2)$	$S \to f_2[A, B]$ $f_2[(x_1), (y_1, y_2)] = (y_1 x_1 y_2)$
$A \to \theta_a$ $\theta_a = (\varepsilon)$	$B \to \theta_b$ $\theta_b = (\varepsilon, \varepsilon)$	
$L_{\mathcal{G}}(A)$ $= \{(a^m) \mid m \in \mathbb{N}\}$	$L_{\mathcal{G}}(B)$ $= \{(b^n, b^n) \mid n \in \mathbb{N}\}$	$L_{\mathcal{G}}(S)$ $= \{(b^n a^m b^n) \mid m, n \in \mathbb{N}\}$

Table 2. Rewriting rules, functions and derivations of Example 2

they are only concatenating the components of their arguments with constant strings in the terminal alphabet. In addition, all tuples that can be derived from A are 1-tuples and all tuples that can be derived from B are 2-tuples.

The rewriting rule for the nonterminal S in this example is different from the rewriting rule for S in Example 1 because the function f contained in the other example did not satisfy the third condition of Definition 4. The function f_2, however, satisfies that condition because it only concatenates the components of its arguments.

The set of all m-multiple context-free grammars is defined as follows.

Definition 6. $\mathcal{L}_{m-MCFL} = \{L | \textit{there exists an m-MCFG } \mathcal{G} \textit{ such that } L = L(\mathcal{G})\}$ *is the set of all m-multiple context-free languages.*

3 Expressivity of m-multiple context-free grammars

The class of m-multiple context-free languages forms a hierarchy, i.e. for all $m \in \mathbb{N} \setminus \{0\}$ it holds that $\mathcal{L}_{m-MCFL} \subsetneq \mathcal{L}_{(m+1)-MCFL}$.

There is a pumping lemma for m-multiple context-free languages. It can be used to show that a language L is not an m-multiple context-free language, i.e. there is no m-MCFG that generates L.

Lemma 1. *For any m-multiple context-free language L which is an infinite set, there exists a word $z \in L$ and strings $u_j \in T^*(1 \leq j \leq m+1)$ and $v_j, w_j, s_j \in T^*(1 \leq j \leq m)$ such that*

1. $z = u_1 v_1 w_1 s_1 u_2 v_2 w_2 s_2 \dots u_m v_m w_m s_m u_{m+1}$
2. $\sum_{j=1}^{m} |v_j s_j| > 0$
3. *for every nonnegative integer i,*
   $z_i = u_1 v_1^i w_1 s_1^i u_2 v_2^i w_2 s_2^i \dots u_m v_m^i w_m s_m^i u_{m+1} \in L$

A proof for this lemma can be found in [1]

The idea behind the pumping lemma for m-multiple context-free languages is similar to the idea of the pumping lemma for context-free languages. In the case of context-free languages, a word can be split such that exactly two pieces of the word can be pumped such that the resulting word is always contained in

the language. With the pumping lemma for m-multiple context-free languages it is possible to pump not only two pieces of a word, but $2m$ pieces.

With the pumping lemma for m-multiple context-free grammars, it is easy to show that the language $L_{2m+2} = \{a_1^n a_2^n a_3^n \dots a_{2m+2}^n | n \geq 1\}$ is not an m-multiple context-free language. However, there is an (m+1)-multiple context-free grammar that generates L_{2m+2} and therefore L_{2m+2} is an (m+1)-multiple context-free language.

Lemma 2. *For all $m \geq 1$, the language $L_{2m+2} = \{a_1^n a_2^n a_3^n \dots a_{2m+2}^n | n \geq 1\}$ is not an m-multiple context-free language.*

Proof. Assume that $L_{2m+2} = \{a_1^n a_2^n a_3^n \dots a_{2m+2}^n | n \geq 1\}$ is an m-MCFL. According to the pumping lemma, there exists a word $z = a_1^\eta a_2^\eta a_3^\eta \dots a_{2m+2}^\eta \in L_{2m+2}$ and strings $u_j \in T^*(1 \leq j \leq m+1)$ and $v_j, w_j, s_j \in T^*(1 \leq j \leq m)$ such that

$$z = u_1 v_1 w_1 s_1 u_2 v_2 w_2 s_2 \dots u_m v_m w_m s_m u_{m+1}$$

and

$$z_2 = u_1 v_1^2 w_1 s_1^2 u_2 v_2^2 w_2 s_2^2 \dots u_m v_m^2 w_m s_m^2 u_{m+1} \in L.$$

Case 1. Now assume that there is some $1 \leq j \leq m$ and some $1 \leq k \leq 2m+1$ such that v_j or s_j contains $a_k a_{k+1}$ as a substring. Then, z_2 will contain $a_k a_{k+1}$ twice as a substring. Therefore $z_i \notin L_{2m+2}$.
Case 2. Now assume that for all $1 \leq j \leq m$ neither v_j nor s_j contains $a_k a_{k+1}$ as a substring. Therefore, for there exists an $h(1 \leq h \leq 2m+2)$ such that the terminal symbol a_h does not occur in $v_1 s_1 v_2 s_2 \dots v_m s_m$. However, since $\sum_{j=1}^{m} |v_j s_j| > 0$ there exists a p $(1 \leq p \leq 2m+2)$ such that the terminal symbol a_p occurs in $v_1 s_1 v_2 s_2 \dots v_m s_m$. Therefore, the terminal symbol a_h will appear exactly η times in z_2 but a_p will appear at least $\eta+1$ times in z_2. Hence, $z_2 \notin L$.

This is a contradiction, and therefore L_{2m+2} is not an m-multiple context-free language.

However, the language L_{2m+2} is an (m+1)-multiple context-free language because there is an (m+1)-multiple context-free grammar that generates L_{2m+2}.

Lemma 3. *For all $m \geq 1$, the language $L_{2m+2} = \{a_1^n a_2^n a_3^n \dots a_{2m+2}^n | n \geq 1\}$ is an (m+1)-multiple context-free language.*

Proof. The following grammar $\mathcal{G}$ is an (m+1)-MCFG that generates L_{2m+2}:
$\mathcal{G} = (\{S, X\}, \bigcup_{i=1}^{m+1} (T^*)^i, \{f, g, \theta\}, \{S \to f[X], X \to g[X], X \to \theta\}, S)$ where

- $f[(x_1, x_2, \dots, x_{m+1})] = x_1 x_2 \dots x_{m+1}$
- $g[(x_1, x_2, \dots, x_{m+1})] = (a_1 x_1 a_2, a_3 x_2 a_4, \dots, a_{2m+1} x_{m+1} a_{2m+2})$
- $\theta = (a_1 a_2, a_3 a_4, \dots, a_{2m+2})$

The following corollary follows from Lemma 2 and Lemma 3.

Corollary 1. *For all $m \geq 1$, $\mathcal{L}_{m-MCFL} \subsetneq \mathcal{L}_{(m+1)-MCFL}$.*

Therefore, the class of m-multiple context-free languages forms a hierarchy and for all $m \in \mathbb{N} \setminus \{0\}$ the expressivity of m-MCFG's is strictly weaker than the expressivity of (m+1)-MCFG's.

4 Tree adjoining languages and m-MCFL's

Tree adjoining grammar is a grammar formalism that deals with trees and an adjoining operation. Formally, a tree adjoining grammar is defined as follows.

Definition 7. *A tree adjoining grammar (TAG) is a 4-tuple* $\mathcal{G} = (N, T, I, A)$.

- N *is a finite set of nonterminal symbols.*
- T *is a finite set of terminal symbols.*
- I *is a finite set of initial trees.*
- A *is a finite set of auxiliary trees.*

In an initial tree, each leaf node is labelled with a terminal symbol and all other nodes are labelled with a nonterminal symbol. In an auxiliary tree, each leaf node except one is labelled with a terminal symbol and all other nodes are labelled with a nonterminal symbol. The leaf node of an auxiliary tree that is labelled with a nonterminal symbol is called foot node. The foot node of an auxiliary tree β *is always labelled with the same symbol as the root node of* β.

Trees can be manipulated by the adjoining operation which is defined as follows.

Definition 8. *Let* $\mathcal{G} = (N, T, I, A)$ *be a TAG and let* α *be a tree such that each leaf node of* α *is labelled with a terminal symbol and each other node of* α *is labelled with a nonterminal symbol. Let* v *be a node in* α*, which is labelled with a nonterminal symbol* $X \in N$*, let* γ *be the subtree in* α *which is dominated by* v *and let* $\beta \in A$ *be an auxiliary tree whose root node is labelled with* $X \in N$[1]*. Let* β' *be the tree that is obtained by replacing the foot node of* β *with the tree* γ.

The tree β *is adjoined to the node* v *in* α *by replacing the subtree* γ *in* α *with the tree* β'.

Figure 1 shows how an auxiliary tree whose frontier is $\alpha_1 A \alpha_2$ is adjoined in a tree whose frontier is $\sigma_1\sigma_2\sigma_3$ to the node that dominates the subtree whose frontier is σ_2 yielding a tree whose frontier is $\sigma_1\alpha_1\sigma_2\alpha_2\sigma_3$.

Definition 9. *Let* $\mathcal{G} = (N, T, I, A)$ *be a tree-adjoining grammar. A tree* τ *can be derived in* $\mathcal{G}$ *if and only if it can be obtained by finitely many adjoining operations, starting with an initial tree in* I*. The tree-adjoining language (TAL) generated by* $\mathcal{G}$ *is defined as the set that contains a string* σ *if and only if a tree* τ *whose frontier is* σ *can be derived in* $\mathcal{G}$*. The TAL generated by* $\mathcal{G}$ *is denoted by* $L(\mathcal{G})$.

Lemma 4. *The set of all tree-adjoining languages* $\mathcal{L}_{TAL}$ *is properly included in* $\mathcal{L}_{2-MCFL}$*, i.e.* $\mathcal{L}_{TAL} \subsetneq \mathcal{L}_{2-MCFL}$.

[1] By Definition 7, the foot node of β is labelled with the same nonterminal symbol X.

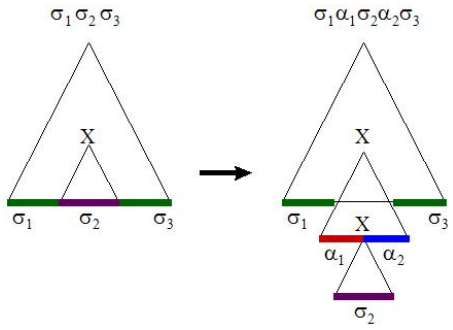


Fig. 1. Adjoining

Seki et al. [1] have shown that the set of tree-adjoining languages is equal to the set of head languages and that the set of head languages is properly included in $\mathcal{L}_{2-MCFL}$.

The following subsection deals with the idea of how a TAG can be transformed into a 2-MCFG that generates the same language. However, proofs for correctness and completeness of the transformation algorithm are not given there.

4.1 $\mathcal{L}_{TAL} \subseteq \mathcal{L}_{2-MCFL}$

Every TAG can be transformed into a 2-MCFG that generates the same language. The idea behind that is to add a rewriting rule and a corresponding function to P and F respectively for every initial tree in I and every auxiliary tree in A.

In order to simplify the transformation, we assume that the TAG is in the following normal form.

Lemma 5. *Every tree-adjoining grammar $\mathcal{G} = (N, T, I, A)$ where $S \notin N$ can be transformed into an equivalent tree-adjoining-grammar $\mathcal{G}' = (N \cup \{S\}, T, I', A')$ that satisfies the following conditions:*

1. *The root of each initial tree in I' is labelled with S.*
2. *Every node that is not the root node of an initial tree is labelled with a symbol other than S.*
3. *No auxiliary tree is adjoinable at the root node of any tree.*
4. *No auxiliary tree is adjoinable at the foot node of any tree.*

Let v be an inner node[2] labelled with a nonterminal symbol $X \in N \setminus \{S\}$ where an auxiliary tree can be adjoined. Assume that the frontier of the subtree is $\sigma_2 \in T^*$ and that the frontier of the entire tree is $\sigma_1\sigma_2\sigma_3$ $(\sigma_1, \sigma_3 \in T^*)$. When

[2] In this context, a node is called inner node if it is labelled with a nonterminal symbol and if it is neither the root node nor the foot node of the tree.

an auxiliary tree whose frontier is $\alpha_1 X \alpha_2$ ($\alpha_1, \alpha_2 \in T^*$) is adjoined in v the frontier of the resulting tree is $\sigma_1\alpha_1\sigma_2\alpha_2\sigma_3$. This idea is illustrated in Figure 1.

Hence, by adjoining a tree in a node v that is labelled with $X \in N \setminus \{S\}$ two substrings will be added to the frontier, one immediately to the left and the other immediately to the right of that part of the frontier that is dominated by v. What substrings can be added solely depends on the nonterminal X. More precisely, $\alpha_1 \in T^*$ and $\alpha_2 \in T^*$ can be added to the left and to the right, respectively, if and only if there is a tree $\tau \in A$ whose frontier is $\alpha_1 X \alpha_2$. Therefore, the fundamental idea for transforming a TAG into a 2-MCFG is that for $X \in N \setminus \{S\}$, the 2-tuple (α_1, α_2) can be derived from X if and only if there is a tree $\tau \in A$ whose frontier is $\alpha_1 X \alpha_2$.

The transformation of a tree-adjoining grammar $\mathcal{G}_{TAG} = (N, T, I, A)$ into a 2-MCFG $\mathcal{G}_{2-MCFG} = (N, O, F, P, S)$ which generates the same language as $\mathcal{G}_{TAG}$ works as follows:

1. The set of nonterminals in $\mathcal{G}_{2-MCFG}$ is the same as the set of nonterminals in $\mathcal{G}_{TAG}$. S, which is the label of the root nodes of the initial trees in I is the start symbol in $\mathcal{G}_{2-MCFG}$. The set of tuples is set to $O = \bigcup_{i=1}^{2} (T^*)^i$.
2. For every initial tree $\alpha \in I$ do the following.
   (a) Let $\{v_1, v_2, \ldots, v_q\}$ be the inner nodes of α.
   (b) Add the function

   $$f : (T^*, T^*)^q \to (T^*), [(l_1, r_1), (l_2, r_2), \ldots, (l_q, r_q)] \mapsto \sigma$$

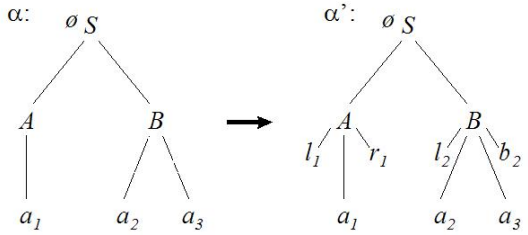
   where σ is the frontier of the tree α' to F. α' is computed as follows. First, let $\alpha' = \alpha$. Then, for each inner node v_i in α'
   - add l_i as the leftmost child to the node v_i and
   - add r_i as the rightmost child to the node v_i.

   (c) For $1 \leq i \leq q$, let $B^{(i)}$ be the nonterminal symbol at the node v_i. Add the rule $S \to f[B^{(1)}, B^{(2)}, \ldots B^{(q)}]$ to P.
3. For every auxiliary tree $\beta \in A$ do the following.
   (a) Let $X \in N$ be the nonterminal at the root node and let $\{v_1, v_2, \ldots, v_q\}$ be the inner nodes of β.
   (b) Add the function

   $$f : (T^*, T^*)^q \to (T^*, T^*), [(l_1, r_1), (l_2, r_2), \ldots, (l_q, r_q)] \mapsto (\sigma_1, \sigma_2)$$

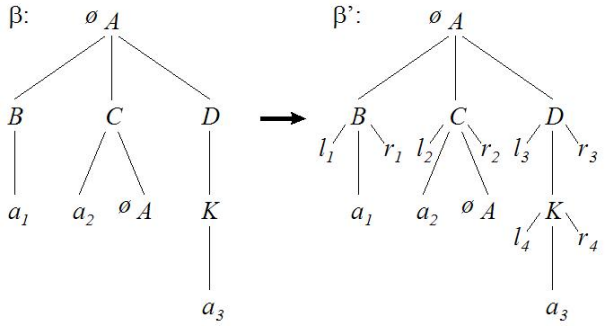
   where $\sigma_1 X \sigma_2$ is the frontier of the tree β' to F. β' is computed as follows. First, let $\beta' = \beta$. Then, for each inner node v_i in β'
   - add l_i as the leftmost child to the node v_i and
   - add r_i as the rightmost child to the node v_i.

   (c) For $1 \leq i \leq q$, let $B^{(i)}$ be the nonterminal symbol at the node v_i. Add the rule $X \to f[B^{(1)}, B^{(2)}, \ldots B^{(q)}]$ to P.
4. Add $\theta = (\varepsilon, \varepsilon)$ to F. For every nonterminal $X \in N \setminus \{S\}$, add the rule $X \to \theta$ to P.



– add $S \rightarrow f[A, B]$ to P
– add $f : (T^*, T^*) \times (T^*, T^*) \rightarrow (T^*), [(l_1, r_1), (l_2, r_2)] \mapsto (l_1 a_1 r_1 l_2 a_2 a_3 r_2)$ to F.

Fig. 2. Transformation of an initial tree into a rewriting rule.



– add $S \rightarrow f[B, C, D, K]$ to P
– add $f : (T^*, T^*)^4 \rightarrow (T^*, T^*)$,
$[(l_1, r_1), (l_2, r_2), (l_3, r_3), (l_4, r_4)] \mapsto (l_1 a_1 r_1 l_2 a_2, r_2 l_3 l_4 a_3 r_4 r_3)$ to F.

Fig. 3. Transformation of an auxiliary tree into a rewriting rule.

In the second step a rewriting rule and a corresponding function are added for each initial tree in I. Figure 2 shows how that rewriting rule and that function are obtained from an example initial tree.

In the third step a rewriting rule and a corresponding function are added for each auxiliary tree in A. Figure 3 shows how that rewriting rule and that function are obtained from an example auxiliary tree.

The fourth step adds terminating rules to P. These are necessary because adjoining is not obligatory.

4.2 $\mathcal{L}_{TAL} \neq \mathcal{L}_{2-MCFL}$

Since every tree adjoining language can be transformed into a 2-MCFG which generates the same language, the set of all tree adjoining languages $\mathcal{L}_{TAL}$ is a

subset of the set of all 2-MCFL's $\mathcal{L}_{2-MCFL}$. To show that there is a 2-MCFL $L \notin \mathcal{L}_{2-TAL}$ a pumping lemma for tree-adjoining languages can be used.

Vijay-Shanker, Weir and Joshi have shown in [2] that the class of modified head languages is equivalent to the class of tree-adjoining languages. Therefore, the pumping lemma for modified head languages shown by Seki et. al in [1] is also applicable for tree-adjoining languages.

Lemma 6. *Let $\#_a(\alpha)$ denote the number of occurrences of the terminal symbol a in the word α. Let L be a tree-adjoining language, such that for a given $n \geq 0$, there exists a word $\alpha \in L$ such that $\#_a(\alpha) \geq n$ holds for every terminal symbol $a \in T$, i.e. every terminal symbol $a \in T$ appears at least n times in α. Then there exists a constant $M \geq 0$ that depends only on L, such that for any $n \geq 0$ there exists a word $z \in L$ that satisfies the following conditions:*

1. *For each $a \in T$, $\#_a(z) \geq n$*
2. *$z = u_1v_1w_1s_1u_2v_2w_2s_2u_3$ such that*
   (a) *$|v_1s_1v_2s_2| \geq 1$*
   (b) *$|u_2| \leq M$*
   (c) *$\forall i \in \mathbb{N} : u_1v_1w_1s_1u_2v_2w_2s_2u_3 \in L$*

The pumping lemma for tree adjoining languages can be used to show that the language $L = \{a^m b^m c^n d^n e^m f^m g^n h^n | m, n \in \mathbb{N}\}$ is not a tree adjoining language.

Lemma 7. *$L = \{a^m b^m c^n d^n e^m f^m g^n h^n | m, n \in \mathbb{N}\}$ is not a tree-adjoining language.*

Proof. Assume that $L = \{a^m b^m c^n d^n e^m f^m g^n h^n | m, n \in \mathbb{N}\}$ is a tree adjoining language and let M be the constant in the pumping lemma. Let $z = a^q b^q c^r d^r e^q f^q g^r h^r = u_1v_1w_1s_1u_2v_2w_2s_2u_3$ where $(q, r > \frac{M}{2})$ and $|v_1s_1v_2s_2| \geq 1$. The condition $\forall i \in \mathbb{N} : u_1v_1^iw_1s_1^iu_2v_2^iw_2s_2^i \in L$ holds only if one of the following two conditions holds:

- $v_1 = a^j$, $s_1 = b^j$, $v_2 = e^j$, $s_2 = f^j$ $(1 \leq j \leq q)$, or
- $v_1 = c^j$, $s_1 = d^j$, $v_2 = g^j$, $s_2 = h^j$ $(1 \leq j \leq r)$.

Since $q, r > \frac{M}{2}$ holds by construction, $|c^r d^r| > M$ and $|e^q f^q| > M$ follows. Therefore, in both cases the condition $|u_2| \leq M$ does not hold. This, however, means that L is not a tree-adjoining language.

Lemma 8. *$L = \{a^m b^m c^n d^n e^m f^m g^n h^n | m, n \in \mathbb{N}\}$ is a 2-MCFL.*

Proof. The following 2-MCFG generates L:

$\mathcal{G} = (\{S, X, Y\}, \bigcup_{i=1}^{m+1}(T^*)^i, \{f, g_x, g_y, \theta\}, \{S \to f[X, Y], X \to g_x[X], X \to \theta, Y \to g_y[Y], Y \to \theta\}, S)$ where

- $f[(x_1, x_2), (y_1, y_2)] = x_1y_1x_2y_2$
- $g_x[(x_1, x_2)] = (ax_1b, ex_2f)$
- $g_y[(y_1, y_2)] = (cy_1d, gx_2h)$
- $\theta = (\varepsilon, \varepsilon)$

The following corollary follows from Lemma 7 and Lemma 8.

Corollary 2. *$\mathcal{L}_{TAL} \neq \mathcal{L}_{2-MCFL}$.*

5 Parsing complexity of m-MCFG's

According to Shieber, Schabes and Pereira [3], parsing can be seen as a deductive process in which inference rules are used to derive statements about the grammatical status of strings from other such statements. This section will give an overview of deductive parsing of MCFG's.

The inference rules become easier, when the grammar is in the normal form that is described in Lemma 9.

Lemma 9. *For every m-MCFG $\mathcal{G} = N, O, F, P, S$ there exists an m-MCFG $G' = N', O', F', P', S'$ such that $L(\mathcal{G}') = L(\mathcal{G})$ and $\mathcal{G}'$ satisfies the following conditions.*

1. *Every terminating rule has the form $A \to (a)$, where $a \in T$ is a terminal symbol.*
2. *For every nonterminal symbol $A \in N' \backslash \{S'\}$ and every tuple $(\alpha_1, \ldots, \alpha_{d(A)}) \in L_{\mathcal{G}'}(A)$ it holds that for all i $(1 \leq i \leq d(A))$, $a_i \neq \varepsilon$, i.e. a tuple that can be derived from a nonterminal which is not the start symbol does not contain any empty component.*
3. *The start symbol S' does not appear in the right-hand side of any rewriting rule in P'.*
4. *For every function $f \in F'$ every component of each argument appears exactly once in the resulting tuple, i.e. ϕ_f as defined in Definition 4 is bijective.*
5. *For every nonterminating rule $A \to f[B^{(1)}, B^{(2)}, \ldots, B^{(q)}] \in P'$, f is defined as concatenation of its arguments, i.e. each constant string $\alpha_{(h,k)}$ as defined in Definition 4 is the empty string.*

A proof can be found in [1].

However, unlike in the Chomsky normal form, typically it can not be achieved that the number of nonterminals in the right-hand side of every production is limited to two. This is the reason why the upper bound for the time needed to decide whether or not a word is contained in the language generated by the m-MCFG $\mathcal{G}$ depends on the functions contained in $\mathcal{G}$ and not only on m [1].

The fundamental idea for parsing of m-MCFG's is similar to bottom-up CKY parsing of context-free grammars. For an m-MCFG $\mathcal{G} = (N, O, F, P, S)$ which is in the normal form described in Lemma 9 and a word $w = w_1 w_2 w_3 \ldots w_n$ parsing works as follows.

The items of the inference rules are of the form $(l_1, r_1, l_2, r_2, \ldots, l_{d(A)}, r_{d(A)}, A)$ and state that the tuple $(w_{l_1} \ldots w_{r_1}, w_{l_2} \ldots w_{r_2}, \ldots, w_{l_{d(A)}} \ldots w_{r_{d(A)}})$ can be derived from $A \in N$ in $\mathcal{G}$. The string w is generated by the grammar $\mathcal{G}$ if and only if the item $(1, n, S)$ is deducible because that item asserts that $w_1 \ldots w_n \in L_{\mathcal{G}}(S) = L(\mathcal{G})$.

If the m-MCFG is in the normal form described in Lemma 9 the terminating rewriting rules can be realized by the following inference rule:

$$\frac{}{(i, i, T)} \; T \to (w_i)$$

The nonterminating rewriting rules can be realized by the following inference rule:

$$\frac{\begin{array}{c}(l_{(1,1)}, r_{(1,1)}, \ldots, l_{(1,d_1(f))}, r_{(1,d_1(f))}, B^{(1)}) \\ (l_{(2,1)}, r_{(2,1)}, \ldots, l_{(2,d_2(f))}, r_{(2,d_2(f))}, B^{(2)}) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ (l_{(a(f),1)}, r_{(a(f),1)}, \ldots, l_{(a(f),d_{a(f)}(f))}, r_{(a(f),d_{a(f)}(f))}, B^{(a(f))})\end{array}}{(\lambda_1, \rho_1, \ldots, \lambda_{r(f)}, \rho_{r(f)}, A)}\ \mathrm{c}$$

where c consists of the following four constraints (with ϕ_f and v_i ($1 \leq i \leq r(f)$) as defined in Definition 4):

1. $A \to f[B^{(1)}, B^{(2)}, \ldots, B^{(a(f))}]$
2. for $1 \leq i \leq r(f)$: $\lambda_i = \phi_f(i, 1)$
3. for $1 \leq i \leq r(f)$: $\rho_i = \phi_f(i, v_i)$
4. for $1 \leq i \leq r(f)$, $1 \leq i < v_i$: $l_{\phi_f(i,j+1)} = r_{\phi_f(i,j)} + 1$

The second constraint states that the left end of each component of the resulting tuple must be the same as the left end of the corresponding argument tuple component that appears leftmost in the concatenation. Similarly, the third constraint states that the right end of each component of the resulting tuple must be the same as the right end of the corresponding argument tuple component that appears rightmost in the concatenation. The fourth constraint states that if two components a and b are concatenated next to each other, yielding ab, then the left end of b must be the right end of a plus 1.

The items in the premises of the inference rule contain $2\sum_{i=1}^{a(f)} d_i(f)$ indices. By the fourth constraint, $\sum_{i=1}^{r(f)}(v_i - 1)$ of these indices are bound. The number of free indices in the inference rules therefore is $2\sum_{i=1}^{a(f)} d_i(f) - \sum_{i=1}^{r(f)}(v_i - 1)$, which can be simplified as follows:

$$\begin{aligned}&2\textstyle\sum_{i=1}^{a(f)} d_i(f) - \sum_{i=1}^{r(f)}(v_i - 1) \\ &= 2\textstyle\sum_{i=1}^{a(f)} d_i(f) - (-r(f) + \sum_{i=1}^{r(f)} v_i) \\ &= r(f) + 2\textstyle\sum_{i=1}^{a(f)} d_i(f) - \sum_{i=1}^{a(f)} d_i(f) \\ &= r(f) + \textstyle\sum_{i=1}^{a(f)} d_i(f)\end{aligned}$$

The second equation holds because ϕ_f is bijective.

Therefore, for an m-MCFG $\mathcal{G} = (N, O, F, P, S)$ which is in the normal form described in Lemma 9 the membership problem for a word α can be decided in $O(|\alpha|^\delta)$ where $\delta = \max_{f \in F}\{d | d = r(f) + \sum_{i=1}^{a(f)} d_i(f)\}$.

6 Summary

Multiple context-free grammar is a weakly context-sensitive subclass of generalized context-free grammar. It deals with tuples of strings and functions that are defined as concatenation of constant strings and components of their argument

tuples . If each of those tuples contains at most m components, the grammar is called an m-multiple context-free grammar.

The class of m-multiple context-free languages forms a hierarchy, that means for all $m \in \mathbb{N} \setminus 0$ the set of all m-multiple context-free languages $\mathcal{L}_{m-MCFL}$ is a strict subset of the set of all (m+1)-multiple context-free languages $\mathcal{L}_{(m+1)-MCFL}$.

In addition, the set of all tree-adjoining languages $\mathcal{L}_{TAL}$ is a strict subset of the set of all 2-multiple context-free grammars $\mathcal{L}_{2-MCFL}$. Therefore, 2-multiple context-free grammars are more expressive than tree-adjoining grammars.

Furthermore, the parsing problem for m-multiple context-free grammars can be decided in polynomial time.

References

1. Seki, Matsumura, Fujii, Kasami: On multiple context-free grammars. TCS: Theoretical Computer Science **88** (1991)
2. Vijay-Shanker, K., Weir, D.J., Joshi, A.K.: Tree adjoining and head wrapping. In: Proceedings of the 11th coference on Computational linguistics, Morristown, NJ, USA, Association for Computational Linguistics (1986) 202–207
3. Shieber, S.M., Schabes, Y., Pereira, F.C.N.: Principles and implementation of deductive parsing. Journal of Logic Programming **24**(1&2) (1995) 3–36