On the relation between Multicomponent Tree Adjoining Grammars with Tree Tuples (TT-MCTAG) and Range Concatenation Grammars (RCG)

Laura Kallmeyer and Yannick Parmentier

Collaborative Research Center 441, University of Tübingen, Germany,
lk@sfs.uni-tuebingen.de, parmenti@sfs.uni-tuebingen.de

Abstract. This paper investigates the relation between TT-MCTAG, a formalism used in computational linguistics, and RCG. RCGs are known to describe exactly the class PTIME; simple RCG even have been shown to be equivalent to linear context-free rewriting systems, i.e., to be mildly context-sensitive. TT-MCTAG has been proposed to model free word order languages. In general, it is NP-complete. In this paper, we will put an additional limitation on the derivations licensed in TT-MCTAG. We show that TT-MCTAG with this additional limitation can be transformed into equivalent simple RCGs. This result is interesting for theoretical reasons (since it shows that TT-MCTAG in this limited form is mildly context-sensitive) and, furthermore, even for practical reasons: We use the proposed transformation from TT-MCTAG to RCG in an actual parser that we have implemented.

1 Introduction

1.1 Tree Adjoining Grammars (TAG)

Tree Adjoining Grammar (TAG, [1]) is a tree-rewriting formalism. A TAG consists of a finite set of trees (elementary trees). The nodes of these trees are labelled with nonterminals and terminals (terminals only label leaf nodes). Starting from the elementary trees, larger trees are derived by substitution (replacing a leaf with a new tree) and adjunction (replacing an internal node with a new tree). In case of an adjunction, the tree being adjoined has exactly one leaf that is marked as the foot node (marked with an asterisk). Such a tree is called an *auxiliary* tree. When adjoining it to a node n, in the resulting tree, the subtree with root n from the old tree is attached to the foot node of the auxiliary tree. Non-auxiliary elementary trees are called *initial* trees. A derivation starts with an initial tree. In a final derived tree, all leaves must have terminal labels. For a sample derivation see Fig. 1.

Definition 1 (Tree Adjoining Grammar)
A Tree Adjoining Grammar *(TAG) is a tuple* $G = \langle I, A, N, T \rangle$ *with*

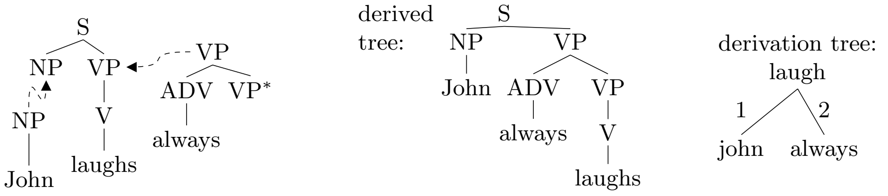


Fig. 1. TAG derivation for *John always laughs*

- *N and T being disjoint finite sets, the nonterminals and terminals*
- *I being a finite set of initial trees with nonterminals N and terminals T, and*
- *A being a finite set of auxiliary trees with nonterminals N and terminals T.*

The internal nodes in $I \cup A$ can be marked as OA (obligatory adjunction) and NA (null adjunction, i.e., no adjunction allowed).

Definition 2 (TAG derivation and tree language) *Let $G = \langle I, A, N, T \rangle$ be a TAG. Let γ and γ' be finite trees.*

- *$\gamma \Rightarrow \gamma'$ in G iff there is a node position p and a tree γ_0' that is either elementary or derived from some elementary tree such that $\gamma' = \gamma[p, \gamma_0']$*[1]*.*
  $\overset{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$.
- *The tree language of G is $L_T(G) = \{\gamma \mid \alpha \overset{*}{\Rightarrow} \gamma$ for some $\alpha \in I$, all leaves in γ have terminal labels and there are no remaining OA nodes in $\gamma\}$.*
  The string language $L(G)$ contains all yields of trees from the tree language.

TAG derivations are represented by derivation trees (unordered trees) that record the history of how the elementary trees are put together. A derived tree is the result of carrying out the substitutions and adjunctions, i.e., the derivation tree describes uniquely the derived tree. Each edge in a derivation tree stands for an adjunction or a substitution. The edges are labelled with Gorn addresses[2]. E.g., the derivation tree in Fig. 1 indicates that the elementary tree for *John* is substituted for the node at address 1 and *always* is adjoined at node address 2 (the fact that the former is an adjunction, the latter a substitution can be inferred from the fact that the node at address 1 is a leaf that is no foot node while the node at address 2 is an internal node).

Definition 3 (TAG derivation tree) *Let $G = \langle I, A, N, T \rangle$ be a TAG. Let γ be a tree derived as follows in G:*

$\gamma = \gamma_0[p_1, \gamma_1] \dots [p_k, \gamma_k]$ where γ_0 is an instance of an elementary tree and the substitutions/adjunctions of the $\gamma_1, \dots, \gamma_k$ are all the substitutions/adjunctions to γ_0 that are performed to derive γ.

[1] For trees $\gamma, \gamma_1, \dots, \gamma_n$ and pairwise different node positions $p_1, \dots, p_n$ in γ, $\gamma[p_1, \gamma_1] \dots [p_n, \gamma_n]$ denotes the result of subsequently substituting/adjoining the $\gamma_1, \dots, \gamma_n$ to the nodes in γ with addresses $p_1, \dots, p_n$ respectively.

[2] The root address is ϵ, and the jth child of a node with address p has address pj.

Then the corresponding derivation tree has a root labelled with γ_0 that has k daughters. The edges from γ_0 to these daughters are labelled with $p_1, \dots, p_k$, and the daughters are the derivation trees for the derivations of $\gamma_1, \dots, \gamma_k$.

1.2 Range Concatenation Grammars (RCG)

This section defines RCGs [2, 3].

Definition 4 (Range Concatenation Grammar) *A* positive Range Concatenation Grammar *is a tuple $G = \langle N, T, V, S, P \rangle$ such that*

- *N is a finite set of predicates, each with a fixed arity;*
- *T and V are disjoint alphabets of terminals and of variables;*
- *$S \in N$ is the start predicate, a predicate of arity 1;*
- *P is a finite set of* clauses *$A_0(x_{01}, \dots, x_{0a_0}) \to \epsilon$, or $A_0(x_{01}, \dots, x_{0a_0}) \to A_1(x_{11}, \dots, x_{1a_1}) \dots A_n(x_{n1}, \dots, x_{na_n})$ with $n \geq 1$ and $A_i \in N, x_{ij} \in (T \cup V)^*$ and a_i being the arity of A_i.*

Since throughout the paper, we use only positive RCGs, whenever we say "RCG", we actually mean "positive RCG"[3]. An RCG with maximal predicate arity n is called an RCG of arity n.

When applying a clause with respect to a string $w = t_1 \dots t_n$, the arguments of the predicates are instantiated with substrings of w, more precisely with the corresponding ranges. A range $\langle i, j \rangle$ with $0 \leq i < j \leq n$ corresponds to the substring between positions i and j, i.e., to the substring $t_{i+1} \dots t_j$. If $i = j$, then $\langle i, j \rangle$ corresponds to the empty string ϵ. If $i > j$, then $\langle i, j \rangle$ is undefined.

Definition 5 *For a given clause, an* instantiation *with respect to a string $w = t_1 \dots t_n$ consists of a function $f : \{t' \,|\, t'$ is an occurrence of some $t \in T$ in the clause$\} \cup V \to \{\langle i, j \rangle \,|\, i \leq j, i, j \in \mathbb{N}\}$ such that*

a) for all occurrences t' of a $t \in T$ in the clause: $f(t') := \langle i, i+1 \rangle$ for some $i, 0 \leq i < n$ such that $t_i = t$,
b) for all $v \in V$: $f(v) = \langle j, k \rangle$ for some $0 \leq j \leq k \leq n$, and
c) if consecutive variables and occurrences of terminals in an argument in the clause are mapped to $\langle i_1, j_1 \rangle, \dots, \langle i_k, j_k \rangle$ for some k, then $j_m = i_{m+1}$ for $1 \leq m < k$. By definition, we then state that f maps the whole argument to $\langle i_1, j_k \rangle$.

The derivation relation is defined as follows:

Definition 6 (RCG derivation and string language)

[3] The negative variant allows for negative predicate calls of the form $\overline{A(\alpha_1, \dots, \alpha_n)}$. Such a predicate is meant to recognize the complement language of its positive counterpart. See [3].

- *For a predicate A of arity k, a clause $A(\dots) \rightarrow \dots$, and ranges $\langle i_1, j_1\rangle, \dots, \langle i_k, j_k\rangle$ with respect to a given w: if there is a instantiation of this clause with left-hand-side $A(\langle i_1, j_1\rangle, \dots, \langle i_k, j_k\rangle)$, then in one derivation step ($\dots \Rightarrow \dots$) $A(\langle i_1, j_1\rangle, \dots, \langle i_i, j_k\rangle)$ can be replaced with the right-hand side of this instantiation. $\overset{*}{\Rightarrow}$ is the reflexive transitive closure of $\Rightarrow$.*
- *The language of an RCG G is*
  $L(G) = \{w \mid S(\langle 0, |w|\rangle) \overset{*}{\Rightarrow} \epsilon$ *with respect to* $w\}$.

For illustration, consider the RCG $G = \langle\{S, A, B\}, \{a, b\}, \{X, Y, Z\}, S, P\rangle$ with $P = \{S(X\,Y\,Z) \rightarrow A(X, Z)\,B(Y), A(a\,X, a\,Y) \rightarrow A(X, Y), B(b\,X) \rightarrow B(X), A(\epsilon, \epsilon) \rightarrow \epsilon, B(\epsilon) \rightarrow \epsilon\}$.

$L(G) = \{a^n b^k a^n \mid k, n \in \mathbb{N}\}$. Take $w = aabaa$. The derivation starts with $S(\langle 0, 5\rangle)$. First we apply the following clause instantiation:

$$\begin{tikzcd}
S(X \arrow[d] & Y \arrow[d] & Z) \arrow[d] & \rightarrow A(X, \arrow[d] & Z) \arrow[d] & B(Y) \arrow[d] \\
\langle 0,2\rangle & \langle 2,3\rangle & \langle 3,5\rangle & \langle 0,2\rangle & \langle 3,5\rangle & \langle 2,3\rangle \\
aa & b & aa & aa & aa & b
\end{tikzcd}$$

With this instantiation, $S(\langle 0, 5\rangle) \Rightarrow A(\langle 0, 2\rangle, \langle 3, 5\rangle)B(\langle 2, 3\rangle)$. Then

$$\begin{tikzcd}
B(b \arrow[d] & X) \arrow[d] & \rightarrow B(X) \arrow[d] & \\
\langle 2,3\rangle & \langle 3,3\rangle & \langle 3,3\rangle & \text{and } B(\epsilon) \rightarrow \epsilon \\
b & \epsilon & \epsilon &
\end{tikzcd}$$

lead to $A(\langle 0, 2\rangle, \langle 3, 5\rangle)B(\langle 2, 3\rangle) \Rightarrow A(\langle 0, 2\rangle, \langle 3, 5\rangle)B(\langle 3, 3\rangle) \Rightarrow A(\langle 0, 2\rangle, \langle 3, 5\rangle)$.

$$\begin{tikzcd}
A(a \arrow[d] & X \arrow[d] & a \arrow[d] & Y) \arrow[d] & \rightarrow A(X, \arrow[d] & Y) \arrow[d] \\
\langle 0,1\rangle & \langle 1,2\rangle & \langle 3,4\rangle & \langle 4,5\rangle & \langle 1,2\rangle & \langle 4,5\rangle \\
a & a & a & a & a & a
\end{tikzcd}$$

leads to $A(\langle 0, 2\rangle, \langle 3, 5\rangle) \Rightarrow A(\langle 1, 2\rangle, \langle 4, 5\rangle)$. Then

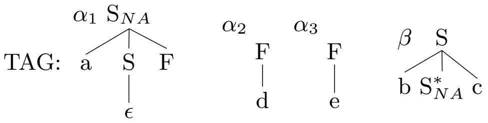
$$\begin{tikzcd}
A(a \arrow[d] & X \arrow[d] & a \arrow[d] & Y) \arrow[d] & \rightarrow A(X, \arrow[d] & Y) \arrow[d] & \\
\langle 1,2\rangle & \langle 2,2\rangle & \langle 4,5\rangle & \langle 5,5\rangle & \langle 2,2\rangle & \langle 5,5\rangle & \text{and } A(\epsilon, \epsilon) \rightarrow \epsilon \\
a & \epsilon & a & \epsilon & \epsilon & \epsilon &
\end{tikzcd}$$

lead to $A(\langle 1, 2\rangle, \langle 4, 5\rangle) \Rightarrow A(\langle 2, 2\rangle, \langle 5, 5\rangle) \Rightarrow \epsilon$

An RCG is called *non-combinatorial* if each of the arguments in the right-hand sides of the productions are single variables. It is called *linear* if no variable appears more than once in the left-hand sides of the productions and no variable appears more than once in the right-hand side of the productions. It is called *non-erasing* if for each production, each variable occurring in the left-hand side occurs also in the right-hand side and vice versa. An RCG is called *simple* if it is non-combinatorial, linear and non-erasing. Simple RCGs and linear context-free rewriting systems (LCFRS, [4]) are equivalent (see [5]). Consequently, simple RCGs are mildly context-sensitive [6].

1.3 From TAG to RCG

Now let us sketch the general idea of the transformation from TAG to RCG, following [7]: The RCG contains predicates $\langle\alpha\rangle(X)$ and $\langle\beta\rangle(L, R)$ for initial and auxiliary trees respectively. X covers the yield of α and all trees added to α, while L and R cover those parts of the yield of β (including all trees added to β) that are to the left and the right of the foot node of β. The clauses in the RCG reduce the argument(s) of these predicates by identifying those parts that come from the elementary tree α/β itself and those parts that come from one of the elementary trees added by substitution or adjunction. A sample TAG with an equivalent RCG is shown in Fig. 2.



Equivalent RCG:
$S(X) \rightarrow \langle\alpha_1\rangle(X) \mid \langle\alpha_2\rangle(X) \mid \langle\alpha_3\rangle(X)$ (every word in the language is the yield of an $\alpha \in I$)
$\langle\alpha_1\rangle(aF) \rightarrow \langle\alpha_2\rangle(F) \mid \langle\alpha_3\rangle(F)$ (the yield of α_1 is a followed by the tree that substitutes at F)
$\langle\alpha_1\rangle(aB_1B_2F) \rightarrow \langle\beta\rangle(B_1, B_2)\langle\alpha_2\rangle(F) \mid \langle\beta\rangle(B_1, B_2)\langle\alpha_3\rangle(F)$ (or β adjoins to S in α; then the yield is a followed by the left part of β, the right part of β and the tree substituted at F)
$\langle\beta\rangle(B_1b, cB_2) \rightarrow \langle\beta\rangle(B_1, B_2)$ (β can adjoin to its root; then the left part is the left part of the adjoined β followed by b; the right part is c followed by the right part of the adjoined β)
$\langle\alpha_2\rangle(d) \rightarrow \epsilon$ $\langle\alpha_3\rangle(e) \rightarrow \epsilon$ $\langle\beta\rangle(b, c) \rightarrow \epsilon$ (the yields of α_2, α_3 and β can be d, e and the pair b (left) and c (right) resp.)

Fig. 2. A sample TAG and an equivalent RCG

2 TT-MCTAG

For a range of linguistic phenomena, multicomponent TAG (MCTAG, [4]) have been proposed. The motivation is the desire to split the contribution of a single lexical item (e.g., a verb and its arguments) into several elementary trees. An MCTAG consists of sets of elementary trees, so-called multicomponents. If a multicomponent is used in a derivation, all its members must be used.

Definition 7 (MCTAG) *A* multicomponent TAG *(MCTAG) is a tuple* $G = \langle I, A, N, T, \mathcal{A}\rangle$ *where* $G_{TAG} := \langle I, A, N, T\rangle$ *is a TAG, and* $\mathcal{A}$ *is a partition of* $I \cup A$*, the set of elementary tree sets.*

The particular type of MCTAG we are concerned with is Tree-Tuple MCTAG with Shared Nodes (TT-MCTAG, [8]). TT-MCTAG were introduced to deal with free word order phenomena in languages such as German. An example is (1) where the argument *es* of *reparieren* precedes the argument *der Mechaniker* of *verspricht* and is therefore not adjacent to the predicate it depends on:

(1) ... dass es der Mechaniker zu reparieren verspricht
... that it the mechanic to repair promises
'... that the mechanic promises to repair it'

A TT-MCTAG is slightly different from standard MCTAG since the elementary tree sets contain two parts: 1. one lexicalized tree γ, marked as the unique *head tree*, and 2. a set of auxiliary trees, the *argument trees*. Such a pair is called a tree tuple. During derivation, the argument trees must either adjoin directly to their head tree or they must be linked by a chain of adjunctions at root nodes to a tree that attaches to the head tree. In other words, in the corresponding TAG derivation tree, the head tree must dominate the auxiliary trees such that all positions on the path between them, except the first one, must be ϵ. This captures the notion of adjunction under node sharing from [9][4].

Definition 8 (TT-MCTAG) *Let $G = \langle I, A, N, T, \mathcal{A}\rangle$ be an MCTAG. G is a TT-MCTAG iff*

1. *every $\mathit{\Gamma} \in \mathcal{A}$ has the form $\{\gamma, \beta_1, \dots, \beta_n\}$ where γ contains at least one leaf with a terminal label, the* head tree*, and $\beta_1, \dots, \beta_n$ are auxiliary trees, the* argument trees*. We write such a set as a tuple $\langle \gamma, \{\beta_1, \dots, \beta_n\}\rangle$.*
2. *A derivation tree D for some $t \in L(\langle I, A, N, T\rangle)$ is licensed as a TAG derivation tree in G iff D satisfies the following conditions (MC) ("multicomponent condition") and (SN-TTL) ("tree-tuple locality with shared nodes"):*
   (a) **(MC)** *There are k pairwise disjoint instances $\mathit{\Gamma}_1, \dots, \mathit{\Gamma}_k$ of elementary tree sets from $\mathcal{A}$ for some $k \geq 1$ such that $\bigcup_{i=1}^{k} \mathit{\Gamma}_i$ is the set of node labels in D.*
   (b) **(SN-TTL)** *for all nodes $n_0, n_1, \dots, n_m$, $m > 1$, in D with labels from the same elementary tree tuple such that n_0 is labelled by the head tree: for all $1 \leq i \leq m$: either $\langle n_0, n_i\rangle \in \mathcal{P}_D$[5] or there are $n_{i,1}, \dots, n_{i,k}$ with auxiliary tree labels such that $n_i = n_{i,k}$, $\langle n_0, n_{i,1}\rangle \in \mathcal{P}_D$ and for $1 \leq j \leq k-1$: $\langle n_{i,j}, n_{i,j+1}\rangle \in \mathcal{P}_D$ where this edge is labelled with ϵ.*

Fig. 3 shows a TT-MCTAG derivation for (1). Here, the NP$_{nom}$ auxiliary tree adjoins directly to *verspricht* (its head) while the NP$_{acc}$ tree adjoins to the root of a tree that adjoins to the root of a tree that adjoins to *reparieren*.

In the general case, the recognition problem for TT-MCTAG is NP-hard [10]. In the following, we define a limitation for TT-MCTAG based on a suggestion from [10]: TT-MCTAG are of rank k if, at any time during the derivation, at most k argument trees depending on higher head trees in the derivation tree are still waiting for adjunction.

[4] The intuition is that if a tree γ' adjoins to some γ, its root in the resulting derived tree somehow belongs both to γ and γ', it is shared by them. A further tree β adjoining to this node can then be considered as adjoining to γ, not only to γ' as in standard TAG. Note that we assume that foot nodes do not allow adjunctions, otherwise node sharing would also apply to them.

[5] For a tree γ, $\mathcal{P}_\gamma$ is the parent relation on the nodes, i.e., $\langle x, y\rangle \in \mathcal{P}_\gamma$ for nodes x, y in γ iff x is the mother of y.

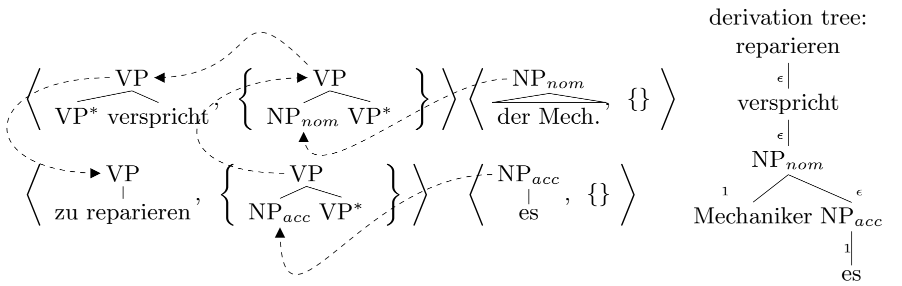


Fig. 3. TT-MCTAG derivation of (1)

Definition 9 (k-TT-MCTAG) *Let $G = \langle I, A, N, T, \mathcal{A}\rangle$ be a TT-MCTAG. G is of rank k (or a k-TT-MCTAG for short) iff for each derivation tree D licenced in G, the following holds:*

(TT-k) *There are no nodes n, $h_0, \dots, h_k$, $a_0, \dots, a_k$ in D such that the label of a_i is an argument tree of the label of h_i and $\langle h_i, n\rangle, \langle n, a_i\rangle \in \mathcal{P}_D^+$ for $0 \le i \le k$.*

With the analyses proposed in [8], that lead to a binary branching structure with a verbal projection line, the linguistic signification of this restriction is roughly that for every VP node on the verbal projection line, at most k NPs can be scrambled over this node. It is hard to say whether such a restriction is empirically valid. Note however, that the number of verbs that allow for non-local scrambling of their arguments is limited. Furthermore, the number of arguments of these verbs is fixed. This indicates that such a limit k actually exists, although it might be motivated rather by semantic and pragmatic reasons than by syntactic reasons.

3 From k-TT-MCTAG to RCG

We construct equivalent simple RCGs for k-TT-MCTAG in a way similar to the RCG construction for TAG. There are predicates $\langle\gamma\rangle$ for the elementary trees (not the tree sets) that characterize the contribution of γ. Recall that each TT-MCTAG is a TAG, a TT-MCTAG derivation is a derivation in the underlying TAG. (This is how we defined TT-MCTAG.) Consequently, we can construct the RCG for the underlying TAG, enrich the predicates in a way that allows to keep track of the "still to adjoin" argument trees and constrain thereby further the RCG clauses. In this case, the yield of a predicate corresponding to a tree γ contains not only γ and its arguments but also arguments of predicates that are higher in the derivation tree and that are adjoined below γ via node sharing[6].

[6] An alternative possibility is to consider only γ and its arguments as the yield of γ. This leads to an RCG with simpler predicate names (the LPAs are not be needed) but with predicates of higher arity since the contribution of γ can be discontinuous: Every argument of a higher head adjoining below γ interrupts the contribution of γ. This construction is much more complex than the one we choose here.

Our construction leads to an RCG of arity 2 with complex predicate names. In order to keep the number of necessary predicates finite, the limit k is crucial.

A predicate $\langle\gamma\rangle$ must encode the set of argument trees that depend on higher head trees and that still need to be adjoined. We call this set the list of pending arguments (LPA). These trees need to either adjoin to the root or to be passed to the LPA of the root-adjoining tree. The LPA is a multiset since we allow for several occurrences of a single tree.

In order to reduce the number of clauses, we distinguish between tree clauses (predicates $\langle\gamma ...\rangle$) and branching clauses (predicates $\langle adj ...\rangle$ and $\langle sub ...\rangle$) following [2]. We therefore have three kinds of predicates:

1. $\langle\gamma, LPA\rangle$ with LPA being the list of pending arguments coming from higher trees (not arguments of γ). This predicate has arity 2 if γ is an auxiliary tree, arity 1 otherwise. $\langle\gamma, LPA\rangle$-clauses distribute the variables for the yields of the trees that substitute or adjoin into γ among corresponding *adj* and *sub* predicates. Furthermore, they pass the LPA to the root-position *adj* predicate and distribute the arguments of γ among the LPAs of all *adj* predicates.
2. $\langle adj, \gamma, dot, LPA\rangle$ as intermediate predicates (of arity 2). Here, LPA contains a) the list of higher args if $dot = \epsilon$, and b) arguments of γ. We assume as a condition that it contains only trees that can be adjoined to *dot* in γ. $\langle adj, \gamma, dot, LPA\rangle$-clauses adjoin a γ' to the *dot* in γ. If $\gamma' \in LPA$, then the new predicate receives $LPA \setminus \{\gamma'\}$. Otherwise, γ' must be a head and LPA is passed unchanged.
3. $\langle sub, \gamma, dot\rangle$ as intermediate predicates (arity 1). $\langle sub, \gamma, dot\rangle$-clauses substitute a γ' into *dot* in γ.

More precisely, the construction goes as follows:

We define the decoration string σ_γ of an elementary tree γ as in [2]: each internal node has two variables L and R and each substitution node has one variable X (L and R represent the left and right parts of the yield of the adjoined tree and X represents the yield of a substituted tree). In a top-down-left-to-right traversal the left variables are collected during the top-down traversal, the terminals and variables of substitution nodes are collected while visiting the leaves and the right variables are collected during bottom-up traversal. Furthermore, while visiting a foot node, a separating "," is inserted. The string obtained in this way is the decoration string.

1. We add a start predicate S and clauses $S(X) \rightarrow \langle\alpha, \emptyset\rangle(X)$ for all $\alpha \in I$.
2. For every $\gamma \in I \cup A$: Let L_p, R_p be the left and right symbols in σ_γ for the node at position p if this is not a substitution node. Let X_p be the symbol for the node at position p if this is a substitution node.
   We assume that $p_1, \ldots, p_k$ are the possible adjunction sites, $p_{k+1}, \ldots, p_l$ the substitution sites in γ. Then the RCG contains all clauses
   $$\langle\gamma, LPA\rangle(\sigma_\gamma) \rightarrow \langle adj, \gamma, p_1, LPA_{p_1}\rangle(L_{p_1}, R_{p_1}) \ldots \langle adj, \gamma, p_k, LPA_{p_k}\rangle(L_{p_k}, R_{p_k})$$
   $$\langle sub, \gamma, p_{k+1}\rangle(X_{p_{k+1}}) \ldots \langle sub, \gamma, p_l\rangle(X_{p_l})$$
   such that

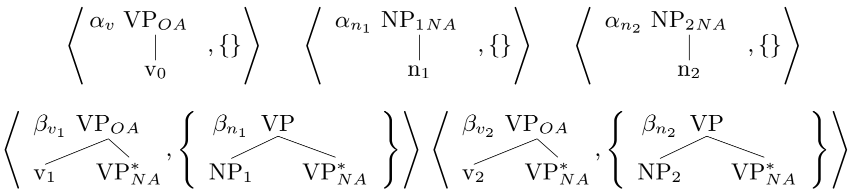


Fig. 4. TT-MCTAG

  - If $LPA \neq \emptyset$, then $\epsilon \in \{p_1, \ldots, p_k\}$ and $LPA \subseteq LPA_\epsilon$, and
  - $\bigcup_{i=0}^{k} LPA_{p_i} = LPA \cup \Gamma(\gamma)$ where $\Gamma(\gamma)$ is either the set of arguments of γ (if γ is a head tree) or (if γ is an argument itself), the empty set.

3. For all predicates $\langle adj, \gamma, dot, LPA\rangle$ the RCG contains all clauses $\langle adj, \gamma, dot, LPA\rangle(L, R) \rightarrow \langle \gamma', LPA'\rangle(L, R)$ such that γ' can be adjoined at position *dot* in γ and
   - either $\gamma' \in LPA$ and $LPA' = LPA \setminus \{\gamma'\}$,
   - or $\gamma' \notin LPA$, γ' is a head (i.e., a head tree), and $LPA' = LPA$.
4. For all predicates $\langle adj, \gamma, dot, \emptyset\rangle$ where *dot* in γ is no OA-node, the RCG contains a clause $\langle adj, \gamma, dot, \emptyset\rangle(\epsilon, \epsilon) \rightarrow \epsilon$.
5. For all predicates $\langle sub, \gamma, dot\rangle$ and all γ' that can be substituted into position *dot* in γ the RCG contains a clause $\langle sub, \gamma, dot\rangle(X) \rightarrow \langle \gamma', \emptyset\rangle(X)$.

As an example consider the TT-MCTAG from Fig. 4. For this TT-MCTAG we obtain (amongst others) the following RCG clauses:

- $\langle \alpha_v, \emptyset\rangle(L\ v_0\ R) \rightarrow \langle adj, \alpha_v, \epsilon, \emptyset\rangle(L, R)$ (only one adjunction at the root, address ϵ)
- $\langle adj, \alpha_v, \epsilon, \emptyset\rangle(L, R) \rightarrow \langle \beta_{v_1}, \emptyset\rangle(L, R) \mid \langle \beta_{v_2}, \emptyset\rangle(L, R)$ (β_{v_1} or β_{v_2} might be adjoined at ϵ in α_v, LPA (here empty) is passed)
- $\langle \beta_{v_1}, \emptyset\rangle(L\ v_1, R) \rightarrow \langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}\}\rangle(L, R)$ (in β_{v_1}, there is only one adjunction site, address ϵ; the argument is passed to the new LPA)
- $\langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}\}\rangle(L, R) \rightarrow$
  $\langle \beta_{n_1}, \emptyset\rangle(L, R) \mid \langle \beta_{v_1}, \{\beta_{n_1}\}\rangle(L, R) \mid \langle \beta_{v_2}, \{\beta_{n_1}\}\rangle(L, R)$ (either β_{n_1} is adjoined and removed from the LPA or another tree (β_{v_1} or β_{v_2}) is adjoined; in this case, the LPA remains)
- $\langle \beta_{v_1}, \{\beta_{n_1}\}\rangle(L\ v_1, R) \rightarrow \langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}, \beta_{n_1}\}\rangle(L, R)$ (again, only one adjunction in β_{v_1}; the argument β_{n_1} is added to the LPA)
- $\langle \beta_{n_1}, \emptyset\rangle(L\ X, R) \rightarrow \langle adj, \beta_{n_1}, \epsilon, \emptyset\rangle(L, R)\ \ \langle sub, \beta_{n_1}, 1, \rangle(X)$ (adjunction to root and substitution to 1 in β_{n_1})
- $\langle adj, \beta_{n_1}, \epsilon, \emptyset\rangle(\epsilon, \epsilon) \rightarrow \epsilon$ (adjunction at root of β_{n_1} not obligatory as long as LPA is empty)
- $\langle sub, \beta_{n_1}, 1, \rangle(X) \rightarrow \langle \alpha_{n_1}, \emptyset\rangle(X)$ (substitution of α_{n_1} at address 1)
- $\langle \alpha_{n_1}, \emptyset\rangle(n_1) \rightarrow \epsilon$ (no adjunctions or substitutions at α_{n_1})
  ...

Take the input word $n_1 n_2 n_1 v_2 v_1 v_1 v_0$. The RCG derivation goes as follows[7]:
$S(n_1\ n_2\ n_1\ v_2\ v_1\ v_1\ v_0) \Rightarrow \langle \alpha_v, \emptyset \rangle (n_1\ n_2\ n_1\ v_2\ v_1\ v_1\ v_0)$

$\Rightarrow \langle adj, \alpha_v, \epsilon, \emptyset \rangle (n_1\ n_2\ n_1\ v_2\ v_1\ v_1, \epsilon)$ (adjunction at ϵ, v_0 is scanned)
$\Rightarrow \langle \beta_{v_1}, \emptyset \rangle (n_1\ n_2\ n_1\ v_2\ v_1\ v_1, \epsilon)$ (β_{v_1} is adjoined)
$\Rightarrow \langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}\} \rangle (n_1\ n_2\ n_1\ v_2\ v_1, \epsilon)$ (adj. at ϵ, v_1 scanned, β_{n_1} put into LPA)
$\Rightarrow \langle \beta_{v_1}, \{\beta_{n_1}\} \rangle (n_1\ n_2\ n_1\ v_2\ v_1, \epsilon)$ (β_{v_1} is adjoined)
$\Rightarrow \langle adj, \beta_{v_1}, \epsilon, \{\beta_{n_1}, \beta_{n_1}\} \rangle (n_1\ n_2\ n_1\ v_2, \epsilon)$ (adj. at ϵ, v_1 scanned, β_{n_1} put into LPA)
$\Rightarrow \langle \beta_{v_2}, \{\beta_{n_1}, \beta_{n_1}\} \rangle (n_1\ n_2\ n_1\ v_2, \epsilon)$ (β_{v_2} is adjoined)
$\Rightarrow \langle adj, \beta_{v_2}, \epsilon, \{\beta_{n_2}, \beta_{n_1}, \beta_{n_1}\} \rangle (n_1\ n_2\ n_1, \epsilon)$ (adj. at ϵ, v_2 scanned, β_{n_2} put into LPA)
$\Rightarrow \langle \beta_{n_1}, \{\beta_{n_2}, \beta_{n_1}\} \rangle (n_1\ n_2\ n_1, \epsilon)$ (β_{n_1} from LPA adjoined)
$\Rightarrow \langle adj, \beta_{n_1}, \epsilon, \{\beta_{n_2}, \beta_{n_1}\} \rangle (n_1\ n_2, \epsilon)\ \ \langle sub, \beta_{n_1}, 1, \rangle (n_1)$ (adj. at ϵ,) (subst. at 1)
$\Rightarrow \langle adj, \beta_{n_1}, \epsilon, \{\beta_{n_2}, \beta_{n_1}\} \rangle (n_1\ n_2, \epsilon)\ \ \langle \alpha_{n_1}, \emptyset \rangle (n_1)$ (subst. of α_{n_1})
$\Rightarrow \langle adj, \beta_{n_1}, \epsilon, \{\beta_{n_2}, \beta_{n_1}\} \rangle (n_1\ n_2, \epsilon)\ \ \epsilon$ (n_1 scanned)
$\Rightarrow \langle \beta_{n_2}, \{\beta_{n_1}\} \rangle (n_1\ n_2, \epsilon)$ (β_{n_2} from LPA adjoined)
$\Rightarrow \langle adj, \beta_{n_2}, \epsilon, \{\beta_{n_1}\} \rangle (n_1, \epsilon)\ \ \langle sub, \beta_{n_2}, 1, \rangle (n_2)$ (adj. at ϵ, subst. at 1)
$\Rightarrow \langle adj, \beta_{n_2}, \epsilon, \{\beta_{n_1}\} \rangle (n_1, \epsilon)\ \ \langle \alpha_{n_2}, \emptyset \rangle (n_2)$ (subst. of α_{n_2})
$\Rightarrow \langle adj, \beta_{n_2}, \epsilon, \{\beta_{n_1}\} \rangle (n_1, \epsilon)\ \ \epsilon$ (n_2 scanned)
$\Rightarrow \langle \beta_{n_1}, \emptyset \rangle (n_1, \epsilon)$ (β_{n_1} from LPA adjoined)
$\overset{*}{\Rightarrow} \langle adj, \beta_{n_1}, \epsilon, \emptyset \rangle (\epsilon, \epsilon)\ \ \langle \alpha_{n_1}, \emptyset \rangle (n_1)$ (subst. of α_{n_1})
$\overset{*}{\Rightarrow} \epsilon$ (scanning of n_1)

This example requires LPAs of maximal cardinality 3, i.e., a 3-TT-MCTAG.

Note that with this construction, the grouping into tree sets gets lost. E.g., in our example, we do not know which of the n_1 came with which of the v_1. However, in our parser we construct the RCG only for the TT-MCTAG of a given input sentence and if the same terminal occurs more than once in the input sentence, we use different occurrences of the corresponding tree tuples.This way, we avoid using the same elementary tree twice and the grouping can be inferred from the tuple identifiers encoded in the names of the trees.

With the above construction the following can be shown:

Theorem 1. *For each k-TT-MCTAG G there is a simple RCG G' with $L(G) = L(G')$*[8]*.*

[7] In this example, we replace the ranges with the corresponding input substrings since this way the example is easier to read.

[8] We suspect that the reverse does not hold. In other words, we suspect that the k-TT-MCTAG languages are properly contained in the set of languages of simple RCGs. An example of a language that is probably not in $\mathcal{L}(k\text{-TT-MCTAG})$ is the double copy language $\{www \mid w \in \{a, b\}^*\}$. The intuition is that, in order to obtain the correct dependencies, the three copies of a terminal (or their substitution slots) must be introduced in a single tree tuple. This means that two of them adjoin via node sharing. But then it is not clear how to avoid getting not only crossing but also other dependencies.

As a corollary, we obtain that the string languages of k-TT-MCTAG are mildly context-sensitive.

To prove the theorem, we introduce TT-RCG derivation trees. These trees are obtained from an RCG derivation by turning the $\langle\gamma, LPA\rangle$ predicates into nodes and the branching predicates into edges.

Definition 10 (TT-RCG derivation tree) *Let G' be an RCG constructed from a k-TT-MCTAG as above. A tree $D_{G'}$ with node and edge labels is a* TT-RCG derivation tree *for G' iff*

- *each node in $D_{G'}$ is labeled with a predicate name $\langle\gamma, LPA\rangle$ and with a sequence of one or (if γ is an auxiliary tree) two $w \in T^*$.*
- *if the root label is $\langle\gamma, LPA\rangle$, then there is a clause $S(X) \rightarrow \langle\gamma, LPA\rangle(X)$;*
- *for every node with label $\langle\gamma, LPA\rangle$ and with l daughters with node labels $\langle\gamma_i, LPA'_i\rangle$ and edge labels dot_i ($1 \leq i \leq l$), there is a $\langle\gamma, LPA\rangle$-clause with $\langle adj...\rangle$ and $\langle sub...\rangle$ predicates on the right-hand side as described in the construction such that*
  - *for all adjunction sites p in γ, $p \notin \{dot_i \,|\, 1 \leq i \leq l\}$: $LPA_p = \emptyset$, and there is a clause $\langle adj, \gamma, p, \emptyset\rangle(\epsilon, \epsilon) \rightarrow \epsilon$*
  - *for all adjunction sites $p = dot_i$ in γ (for some i, $1 \leq i \leq l$): there is a clause $\langle adj, \gamma, dot_i, LPA_p\rangle(L, R) \rightarrow \langle\gamma_i, LPA'_i\rangle(L, R)$*
  - *for all substitution sites $p = dot_i$ in γ (for some i, $1 \leq i \leq l$): $LPA'_i = \emptyset$ and there is a clause $\langle sub, \gamma, dot_i\rangle(X) \rightarrow \langle\gamma_i, \emptyset\rangle(X)$*
- *for every leaf with label $\langle\gamma, LPA\rangle$ and $\langle w\rangle$ (or $\langle w_1, w_2\rangle$ resp.), there is a clause $\langle\gamma, LPA\rangle(w) \rightarrow \epsilon$ (or $\langle\gamma, LPA\rangle(w_1, w_2) \rightarrow \epsilon$ resp.).*
- *the sequences of strings for a mother node are computed from the daughters such that for at least one word w, the clauses leading from the mother to the daughters can be instantiated successfully, assuming an instantiation with an empty range for all variables not passed to one of the daughter predicates.*

Furthermore, we call a TAG derivation tree whose nodes are equipped with the yields of the derivation trees they root (one component for initial trees, two components for auxiliary trees) and the set of arguments they dominate that actually depend on higher head trees a *decorated TAG derivation tree.*

Once these structures are defined, we can prove the correspondence between the decorated TAG derivation trees licensed in the k-TT-MCTAG G and the TT-RCG derivation trees of the RCG G'. More precisely, we show that for each decorated TAG derivation tree in G, there is an isomorphic TT-RCG derivation tree in G' and vice versa. We can show this by an induction on the height of the subtree rooted by a node. (Due to space limitations, we omit the proof here.)

Conclusion

This paper has investigated the relation between two grammar formalisms, TT-MCTAG and RCG. TT-MCTAG is a tree rewriting formalism that allows to adequately model the free word order in certain languages, e.g., German. RCG, on the other hand, is known to have nice formal properties: RCGs in general are polynomially parsable, simple RCGs are even mildly context-sensitive. Furthermore, parsing algorithms for simple RCGs are already available.

In this paper, we have shown how to construct for a given TT-MCTAG with a certain limitation (a so-called k-TT-MCTAG) an equivalent simple RCG. As a formal result, we obtain that the class of string languages generated by k-TT-MCTAG is contained in the class of languages generated by simple RCGs. In particular, k-TT-MCTAG are mildly context-sensitive.

As a practical result, we can use this transformation from k-TT-MCTAG to simple RCG for a 2-step k-TT-MCTAG parser that, in a first step, does the transformation and, in a second step, parses with the RCG obtained from the first step. As we have seen from the correspondence between the two derivation structures, the derivation tree of the k-TT-MCTAG can be retrieved from the RCG parse tree in a straightforward way. We have implemented this within a project that develops a TAG-based grammar for German along with a parser for this grammar[9].

References

1. Joshi, A.K., Schabes, Y.: Tree-Adjoning Grammars. In Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Springer, Berlin (1997) 69–123
2. Boullier, P.: On TAG Parsing. In: TALN 99, 6^e conférence annuelle sur le Traitement Automatique des Langues Naturelles, Cargèse, Corse (1999) 75–84
3. Boullier, P.: Range Concatenation Grammars. In: Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT2000), Trento, Italy (2000) 53–64
4. Weir, D.J.: Characterizing mildly context-sensitive grammar formalisms. PhD thesis, University of Pennsylvania (1988)
5. Boullier, P.: A Proposal for a Natural Language Processing Syntactic Backbone. Technical Report 3342, INRIA (1998)
6. Joshi, A.K.: Tree adjoining grammars: How much contextsensitivity is required ro provide reasonable structural descriptions? In Dowty, D., Karttunen, L., Zwicky, A., eds.: Natural Language Parsing. Cambridge University Press (1985) 206–250
7. Boullier, P.: A Generalization of Mildly Context-Sensitive Formalisms. In: Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+4), University of Pennsylvania, Philadelphia (1998) 17–20
8. Lichte, T.: An MCTAG with Tuples for Coherent Constructions in German. In: Proceedings of the 12th Conference on Formal Grammar 2007, Dublin, Ireland (2007)
9. Kallmeyer, L.: Tree-local multicomponent tree adjoining grammars with shared nodes. Computational Linguistics **31**(2) (2005) 187–225
10. Søgaard, A., Lichte, T., Maier, W.: The complexity of linguistically motivated extensions of tree-adjoining grammar. In: Recent Advances in Natural Language Processing 2007, Borovets, Bulgaria (2007)

[9] See `http://www.sfb441.uni-tuebingen.de/emmy/tulipa`.