Multi-Component Tree Insertion Grammars

Pierre Boullier, Benoît Sagot

Multi-Component Tree Insertion Grammars

Pierre Boullier & Benoît Sagot

Alpage, INRIA Paris-Rocquencourt & Université Paris 7
Domaine de Voluceau — Rocquencourt, BP 105
78153 Le Chesnay Cedex, France
{Pierre.Boullier,Benoit.Sagot}@inria.fr

Abstract. In this paper we introduce a new mildly context sensitive formalism called Multi-Component Tree Insertion Grammar. This formalism is a generalization of Tree Insertion Grammars in the same sense that Multi-Component Tree Adjoining Grammars is a generalization of Tree Adjoining Grammars. We show that this class of grammatical formalisms is equivalent to Multi-Component Tree Adjoining Grammars, and that it also defines a hierarchy of languages whose supplementary formal power between two increasing levels is more gently delivered than the one given by Multi-Component Tree Adjoining Grammars. We show that Multi-Component Tree Insertion Grammars and simple Range Concatenation Grammars are equivalent and we show how to transform a grammar of one type into an equivalent grammar of the other type. Such a transformation gives a method to build efficient parsers for Multi-Component Tree Insertion Languages.

1 Introduction

The notion of mild context-sensitivity (MCS) [1, 2] is an attempt to express the formal power needed to define the syntax of natural languages. However, all incarnations of MCS formalisms are not equivalent. On the one hand, near the bottom of the hierarchy, we find tree adjoining grammars (TAGs) [3], the most popular mild context-sensitive formalism, and some other weakly equivalent formalisms. On the other hand, near the top of the hierarchy, we find Multi-Component Tree Adjoining Grammars (MCTAGs) which are, in turn and among others, equivalent to Linear Context-Free Rewriting Systems (LCFRSs) [4].

In this paper we introduce a new mild context-sensitive formalism, the multi-component tree insertion grammar (MCTIG) which plays, w.r.t. tree insertion grammar TIG the same role as MCTAG plays w.r.t. TAG. We know that TIGs are weakly equivalent to context-free grammars (CFGs) but they allow to build (parse) structures that are not possible to build with CFGs. From a practical point of view we know that TIGs, which may be seen as a restriction of TAGs, are of importance since many usual linguistic constructions do not need the power of TAGs. On the other hand there exist linguistic constructions that cannot be expressed with TAGs. Thus MCTAGs have been introduced. These various formalisms named k-MCTAGs form a strict hierarchy which depends upon an

integer k. TAGs are 1-MCTAGs and the languages of k-MCTAGs (k-MCTALs) are strictly included into $k+1$-MCTALs. However the extra power earned when we use $k+1$-MCTAGs instead of k-MCTAGs is too large. For example let L_i be the counting language for i ($L_i = \{a_1^n \dots a_i^n \mid n \geq 0\}$). It is well known that 1-MCTAGs (i.e., TAGs) can define L_1, L_2, L_3 and L_4 but not L_5. If we want to define L_5 we must use 2-MCTAGs. However, doing that we can also define L_6, L_7 and L_8. We shall see that the power of k-MCTIGs is more balanced.

On the other hand, though being non MCS, there is a very attractive powerful formalism named Range Concatenation Grammar (RCG). Its power comes from the fact that it exactly covers the class PTIME and its attractivity comes from both theoretical and practical considerations. From a theoretical point of view it possesses many closure properties among which the closure by intersection is the most salient. From a practical point of view there exist very efficient polynomial parse time parsers. For example TAGs and MCTAGs have been transformed into equivalent RCGs [5, 6] whose parsers achieved top-of-the-art efficiency [7].

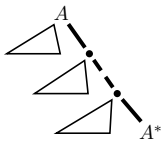
In this paper we define MCTIGs and study its relationships with both MCTAGs and RCGs, thus exhibiting a way to produce an efficient MCTIG parser.

2 Basic notions and notations

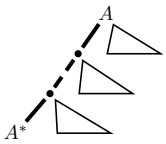
2.1 Tree Insertion Grammars

We suppose that the reader is familiar with Tree Adjoining Grammars (TAGs), as defined in, e.g., [3].

Tree Insertion Grammar (TIG) [8] is a variant of TAG in which the auxiliary tree shapes have a restricted form: wrapping auxiliary trees are prohibited leaving only left and right auxiliary trees.[1] A *left auxiliary tree* (resp. *right auxiliary tree*) is an auxiliary tree in which the foot node is the rightmost (resp. leftmost) leaf node and in which adjunction nodes of its spine only select left auxiliary trees (resp. right auxiliary trees) or empty auxiliary trees.



Left Auxiliary Tree



Right Auxiliary Tree

Empty Auxiliary Tree

.

Thus the set of auxiliary trees $\mathcal{A}$ can be partitioned into three sets $\mathcal{L}$, $\mathcal{R}$ and $\mathcal{E}$, respectively the set of left, right and empty trees.

It is a well known result [8] that TIGs are (weakly) equivalent to CFGs and can be parsed in $\mathcal{O}(n^3)$ time.

[1] In the original definition of TIGs, there exist other differences which are not considered in this article. In particular we disallow simultaneous adjunctions at a single node (they can be performed by the usual adjunction operation), but we allow empty auxiliary trees.

2.2 Multi-Component Tree Adjoining Grammars

Multi-component TAGs (MCTAGs) are an extension of TAGs which was introduced in [9] and later refined in [3], in which the adjunction operation involves a set of auxiliary trees instead of a single auxiliary tree. In a MCTAG, the elementary structures, both initial and auxiliary, instead of being two sets of single trees, consist of two finite sets of (ordered) finite tree sets. In MCTAGs, the adjunction operation of an auxiliary tree set is defined as the simultaneous adjunction of each of its component trees and accounts for a single step in the derivation process. This multi-component adjunction (MCA) operation is defined as follows. All the trees of an auxiliary tree set can be adjoined into distinct nodes (addresses) in a single elementary (initial or auxiliary) tree set. If the maximum cardinality of the elementary tree sets is k, we have a k-MCTAG. Of course, if the cardinality of each tree set is one, a 1-MCTAG is a TAG. In [2], the author has shown that the languages defined by MCTAGs are equal to the languages defined by Linear Context-Free Rewriting Systems (LCFRSs) [4].

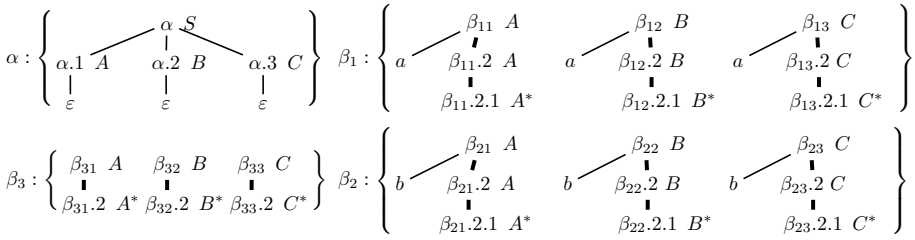
We assume that (multi-component) substitution operations are disallowed and are replaced by MCA operations. Thus, without loss of generality, we assume that initial tree sets are singletons whose root nodes are all labelled by the start symbol S. Moreover, we assume that adjunction is allowed neither at the root nor at the foot of any tree and that MCAs are mandatory on inside nonterminal nodes which are thus called *adjunction nodes*.

We can think of two types of *locality* for MCAs, one type, named *tree locality*, requires that all trees in an auxiliary tree set adjoin to a unique tree of an elementary tree set; the other type, named *set locality*, requires that all trees in an auxiliary tree set adjoin to the same elementary tree set, not necessarily to a unique tree and not necessarily to all the trees in this elementary tree set. We choose to consider the set-local interpretation of the term since it is more general than the tree-local version and is the one equivalent to LCFRS.

Example 1. For example, let G be a 3-MCTAG in which the set of initial tree sets is $\mathcal{I} = \{\{\alpha\}\}$ and the set of auxiliary tree sets is $\mathcal{A} = \{\beta_1, \beta_2, \beta_3\}$, where each tree set β_1, β_2 and β_3 contains three trees: $\beta_1 = \{\beta_{11}, \beta_{12}, \beta_{13}\}$, $\beta_2 = \{\beta_{21}, \beta_{22}, \beta_{23}\}$ and $\beta_3 = \{\beta_{31}, \beta_{32}, \beta_{33}\}$. Its elementary trees are depicted below.[2]

We can easily see that the language defined by G is the non-TAL three-copy language $\{www \mid w \in \{a, b\}^*\}$. Each simultaneous adjunction of an auxiliary tree set β_1 or β_2 in the elementary tree sets α, β_1 or β_2 produces respectively three related a's or three related b's, while the MCA of (the trees in) β_3 terminates the process.

[2] The same denotation is used both for nonterminal nodes and their addresses, while terminal nodes are denoted by their terminal labels or ε. The root node of an elementary tree τ is also denoted by τ. Nonterminal nodes are annotated by their nonterminal labels while terminal nodes have no annotations. In auxiliary trees, foot node labels are marked by an $*$. For example, the root of tree β_{11} is the node β_{11}, whose label is A, and the foot node of this tree is $\beta_{11}.2.1$ and its label is A^*.



2.3 Positive Range Concatenation Grammars

A *positive range concatenation grammar* (PRCG) $G = (N, T, V, P, S)$ is a 5-tuple in which:

- T and V are disjoint alphabets of *terminal symbols* and *variable symbols* respectively.
- N is a non-empty finite set of *predicates* of fixed *arity* (also called *fan-out*). We write $k = arity(A)$ if the arity of the predicate A is k. A predicate A and its arguments is noted $A(\boldsymbol{\alpha})$ with a vector notation s.t. $|\boldsymbol{\alpha}| = k$ and $\boldsymbol{\alpha}[j]$ is its j^{th} argument. An argument is a string in $(V \cup T)^*$.
- S is a distinguished predicate called the *start predicate* (or *axiom*) of arity 1.
- P is a finite set of *clauses*. A clause c is a rewriting rule of the form $A_0(\boldsymbol{\alpha_0}) \rightarrow A_1(\boldsymbol{\alpha_1}) \dots A_r(\boldsymbol{\alpha_r})$ where r, $r \geq 0$ is its *rank*. By definition $c[i] = A_i(\boldsymbol{\alpha_i})$, $0 \leq i \leq r$ where A_i is a predicate together with $\boldsymbol{\alpha_i}$ its arguments and $c[i][j]$ is its j^{th} argument $X_1 \dots X_{n_{ij}}$ (the X_k's are terminal or variable symbols), while $c[i][j][k]$, $0 \leq k \leq n_{ij}$ is a *position*.

For a given clause c, each *subargument occurrence* is denoted by a pair of positions $(c[i][j][k], c[i][j][k'])$ with $k \leq k'$.

Let $w = a_1 \dots a_n$ be an input string in T^*, each occurrence of a substring $a_{l+1} \dots a_u$ is a pair of positions $(w[l], w[u])$ s.t. $0 \leq l \leq u \leq n$ called a *range* and noted $\langle l..u \rangle_w$ or $\langle l..u \rangle$ when w is implicit. In the range $\langle l..u \rangle$, l is its *lower bound* while u is its *upper bound*. If $l = u$, the range $\langle l..u \rangle$ is an *empty* range, it spans an empty substring. If $\rho_1 = \langle l_1..u_1 \rangle$, ... and $\rho_m = \langle l_m..u_m \rangle$ are ranges, the *concatenation* of ρ_1, ..., ρ_m noted $\rho_1 \dots \rho_m$ is the range $\rho = \langle l..u \rangle$ if and only if we have $u_i = l_{i+1}$, $1 \leq i < m$, $l = l_1$ and $u = u_m$.

If $c = A_0(\boldsymbol{\alpha_0}) \rightarrow A_1(\boldsymbol{\alpha_1}) \dots A_r(\boldsymbol{\alpha_r})$ is a clause, each of its subargument occurrence $(c[i][j][k], c[i][j][k'])$ may take a range $\rho = \langle l..u \rangle$ as value, in that case, we say that it is *instantiated* by ρ.

- If the subargument is the empty string (i.e., $k = k'$), ρ is an empty range.
- If the subargument is a terminal symbol (i.e., $k + 1 = k'$ and $X_{k'} \in T$), ρ is such that $l + 1 = u$ and $a_u = X_{k'}$. Note that several occurrences of the same terminal symbol may be instantiated by different ranges.

– If the subargument is a variable symbol (i.e., $k+1 = k'$ and $X_{k'} \in V$), any occurrence $(c[i'][j'][m], c[i'][j'][m'])$ of $X_{k'}$ is instantiated by ρ. Thus, each occurrence of the same variable symbol must be instantiated by the same range.
– If the subargument is the string $X_{k+1} \dots X_{k'}$, ρ is its instantiation if and only if we have $\rho = \rho_{k+1} \dots \rho_{k'}$ in which ρ_{k+1}, …, $\rho_{k'}$ are respectively the instantiations of X_{k+1}, …, $X_{k'}$.

If in c we replace each argument by a valid instantiation, we get an *instantiated clause* noted $A_0(\boldsymbol{\rho_0}) \to A_1(\boldsymbol{\rho_1}) \dots A_r(\boldsymbol{\rho_r})$ in which each $A_i(\boldsymbol{\rho_i})$ is an *instantiated predicate*.

A binary relation called *derive* and noted $\underset{G,w}{\Rightarrow}$ is defined on strings of instantiated predicates. If Γ_1 and Γ_2 are strings of instantiated predicates, we have

$$\Gamma_1\ A_0(\boldsymbol{\rho_0})\ \Gamma_2 \underset{G,w}{\Rightarrow} \Gamma_1\ A_1(\boldsymbol{\rho_1}) \dots A_m(\boldsymbol{\rho_m})\ \Gamma_2$$

if and only if $A_0(\boldsymbol{\rho_0}) \to A_1(\boldsymbol{\rho_1}) \dots\ A_m(\boldsymbol{\rho_m})$ is an instantiated clause.

The *(string) language* of a PRCG G is the set $\mathcal{L}(G) = \{w \mid S(\langle 0..|w|\rangle_w) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon\}$. In other words, an input string $w \in T^*$, $|w| = n$ is a *sentence* of G if and only there exists a *complete derivation* which starts from $S(\langle 0..n\rangle)$ (the instantiation of the start predicate on the whole input text) and leads to the empty string (of instantiated predicates). The *parse forest* of w is the CFG whose axiom is $S(\langle 0..n\rangle)$ and whose productions are the instantiated clauses used in all complete derivations.[3]

We say that the arity of a PRCG is k, and we write k-PRCG, if and only if k is the maximum arity of its predicates ($k = \max_{A \in N} \mathit{arity}(A)$). We say that a k-PRCG is *simple*, we have a simple k-PRCG, if and only if each of its clause is

– *non-combinatorial*: the arguments of its RHS predicates are single variables;
– *non-erasing*: each variable which occur in its LHS (resp. RHS) also occurs in its RHS (resp. LHS);
– *linear*: there are no variables which occur more than once in its LHS and in its RHS.

The subclass of simple PRCGs is of importance since it is MCS and is the one equivalent to LCFRSs.

We say that a simple 2-PRCG clause of the form

$$\phi \to A_1(X_1, X_1') \dots A_p(X_p, X_p')\ B_1(Y_1) \dots B_q(Y_q)$$

in which the LHS is either of the form $B_0(\alpha)$ or $A_0(\alpha_0, \alpha_0')$ with $\alpha = \alpha_0\alpha_0'$ is *well-balanced* if and only if α is a Dyck string w.r.t. the pairs (X_i, X_i') which play the role of parentheses, X_i is an open parenthese while X_i' is a closing parenthese. A simple 2-PRCG is *well-balanced* if its clauses are all well-balanced.

[3] Note that this parse forest has no terminal symbols (its language is the empty string).

3 TAGs and MCTAGs vs. Simple PRCGs

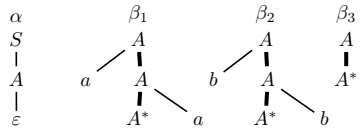
In this Section we briefly present the algorithms of [10, 5] with a slight modification due to our (simplified) vision[4] of TAGs and MCTAGs.

3.1 Transforming TAGs into Simple 2-PRCGs

If we consider an auxiliary tree τ and the way it evolves until no more adjunction is possible, we realize that some properties of the final tree are already known on τ. The yield derived by the part of τ to the left (resp. to the right) of its spine are contiguous terminals and the *left yield* (produced by the left part) lies to the left of the *right yield* in any input string. Thus, for any elementary tree τ consider its m internal adjunction nodes. We decorate each such node η_i with two variables L_{η_i} and R_{η_i} $(1 \leq i \leq m)$ which are supposed to capture respectively the left and right yield of the trees that adjoined at η_i. Each terminal leaf has a single decoration which is its terminal label. Then, during a top-down left-to-right traversal of τ, we collect into a string σ_τ called *decoration string*, all such decorations. If τ is an auxiliary tree, let σ_τ^l and σ_τ^r be the part of σ_τ gathered before and after the traversal of the root of τ. To each elementary tree τ, we associate a simple 2-PRCG clause constructed as follows:

- its LHS is the predicate $S(\sigma_\tau)$ if τ is an initial S-tree or
- its LHS is the predicate $A(\sigma_\tau^l, \sigma_\tau^r)$ if τ is an auxiliary A-tree;
- its RHS is $\psi_1 \dots \psi_i \dots \psi_m$ with $\psi_i = A_i(L_{\eta_i}, R_{\eta_i})$ where A_i is the label of η_i.

Example 2. The following TAG



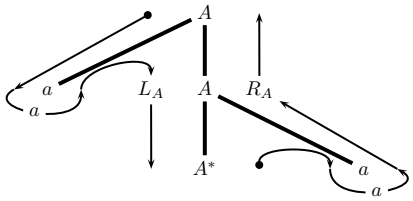
where α is its initial tree and β_1, β_2 and β_3 are its auxiliary trees, defines the language $\{ww \mid w \in \{a, b\}^*\}$ which is translated into the strongly equivalent simple 2-PRCG[5]

$$\begin{array}{ll} S(L_A R_A) & \rightarrow A(L_A, R_A) \\ A(aL_A, aR_A) & \rightarrow A(L_A, R_A) \\ A(bL_A, bR_A) & \rightarrow A(L_A, R_A) \\ A(\varepsilon, \varepsilon) & \rightarrow \varepsilon \end{array}$$

As an example, the arguments of the LHS predicate of the second clause have been gathered during the following traversal of β_1

[4] No substitution, no adjunction neither at the root nor at the foot node and mandatory adjunction on inside nodes.

[5] Since there is no possible confusion, nodes are addressed directly by their labels.



It has been shown in [5] how the simple 2-PRCG generated by the previous algorithm can be turned into a normal form which contains at most two variables (the 2-var form) in each argument.[6] Since the standard parser of [11] has a worst case parse time complexity of $\mathcal{O}(|G|n^{k+v})$ for a k-RCG G in which v is the maximum number of variables per clause, we reach the classical $\mathcal{O}(n^6)$ parse time for TAGs.

3.2 Transforming Set-Local k-MCTAGs into Simple $2k$-PRCGs

The transformation of any set-local k-MCTAG into an equivalent simple $2k$-PRCG is based upon a generalization of the algorithm of Section 3.1. The difficulty is to identify the nodes where MCA operations can take place. This will be done in introducing the notion of *cover*.

Let us consider an elementary tree set γ. For each of its internal nodes η which is a (mandatory) adjunction node, we have to find a tree τ_i (in some auxiliary tree set β) which could be adjoined at η. A mapping noted ξ from each adjunction node of γ to an auxiliary tree τ_i (we have $\tau_i = \xi(\eta)$) in some auxiliary tree set β is called a *cover* (of γ). Note that different adjunction nodes of γ may be associated with trees that come from different tree sets. To be a cover, the total function ξ must fullfilled three constraints for each internal node η:

1. if $\xi(\eta) = \tau_i$, the label of η must be the label of the root of τ_i (only adjunction of A-trees are allowed on A-nodes);
2. if β is the tree set such that $\tau_i \in \beta$, then for each auxiliary tree $\tau_j \in \beta$ there exists an adjunction node of γ such that $\xi(\eta') = \tau_j$ (this constraint is the consequence of the definition of a MCA operation as the simultaneous adjunction of all its component trees in an auxiliary tree set) and
3. if η_1 and η_2 are two distinct nodes of γ s.t. $\xi(\eta_1) = \tau$ and $\xi(\eta_2) = \tau$ for some auxiliary tree τ in some auxiliary tree set β and if τ' is another tree of β then there exist two distinct nodes η'_1 and η'_2 of γ s.t. $\xi(\eta'_1) = \tau'$ and $\xi(\eta'_2) = \tau'$ (this constraint forces trees in an auxiliary tree set to be adjoined the same number of times during any MCA).

Note that empty auxiliary trees have exactly one cover, namely the empty cover.

[6] Such a transformation, which is always possible, is based upon the fact that decoration strings are well-parenthesized (Dyck) string.

Of course, for any given elementary tree set γ its cover is, in the general case, not unique (it may even not exist, and in this case γ can not be used in a derivation). However, for any elementary tree set in any given MCTAG, the number of its covers is bounded. Let *cover*(γ) be the set of all the covers of the elementary tree set γ.

The generation of RCG clauses proceeds as follow. Each elementary tree set γ will yield $|$*cover*(γ)$|$ simple PRCG clauses, one for each cover ξ in *cover*(γ). All these clauses will share the extraction of the decoration strings of γ. If γ is an initial tree set (a singleton), we associate a single decoration string σ_γ, while, if γ is an auxiliary tree set s.t. $|\gamma| = k$, we associate a sequence of $2k$ decoration strings $\sigma^l_{\gamma.1}, \sigma^r_{\gamma.1}, \ldots, \sigma^l_{\gamma.k}, \sigma^r_{\gamma.k}$, the pair $\sigma^l_{\gamma.h}, \sigma^r_{\gamma.h}$ in this sequence being respectively the left and right decoration strings of the (auxiliary) tree of rank h in γ. All $|$*cover*(γ)$|$ simple PRCG clauses will have the same LHS: $S(\sigma_\gamma)$ if γ is an initial tree set or $< \gamma > (\sigma^l_1, \sigma^r_1, \ldots, \sigma^l_k, \sigma^r_k)$ if γ is an auxiliary tree set. Now consider the RHS's of these clauses. Each ξ in *cover*(γ) will produce a different RHS. For a given ξ, let $\beta_1, \ldots, \beta_m$ be the list of auxiliary tree sets that it selects. The RHS of the clause computed for ξ will be of the form $< \beta_1 > (L^1_1, R^1_1 \ldots, L^{|\beta_1|}_1, R^{|\beta_1|}_1) \ldots < \beta_m > (L^1_m, R^1_m \ldots, L^{|\beta_m|}_m, L^{|\beta_m|}_m)$ in which the L's and R's are the left and right variables that occur in the LHS decoration strings and that are associated to the adjunction nodes of γ. For each node η in γ let its two variables be L_η and R_η and let r be the rank of the auxiliary tree $\xi(\eta)$ in the auxiliary tree set β_j. We thus have $L_\eta = L^{\beta_j}_r$ and $R_\eta = R^{\beta_j}_r$.

Example 3. Let us consider the 3-MCTAG of Example 1. The decoration string σ_α of its initial tree set α is $\sigma_\alpha = L_{\alpha.1}R_{\alpha.1}L_{\alpha.2}R_{\alpha.2}L_{\alpha.2}R_{\alpha.2}$. It yields the three clauses

$$\begin{array}{c} S(L_{\alpha.1}R_{\alpha.1}L_{\alpha.2}R_{\alpha.2}L_{\alpha.2}R_{\alpha.2}) \rightarrow < \beta_1 > (L_{\alpha.1}, R_{\alpha.1}, L_{\alpha.2}, R_{\alpha.2}, L_{\alpha.2}, R_{\alpha.2}) \\ S(L_{\alpha.1}R_{\alpha.1}L_{\alpha.2}R_{\alpha.2}L_{\alpha.2}R_{\alpha.2}) \rightarrow < \beta_2 > (L_{\alpha.1}, R_{\alpha.1}, L_{\alpha.2}, R_{\alpha.2}, L_{\alpha.2}, R_{\alpha.2}) \\ S(L_{\alpha.1}R_{\alpha.1}L_{\alpha.2}R_{\alpha.2}L_{\alpha.2}R_{\alpha.2}) \rightarrow < \beta_3 > ((L_{\alpha.1}, R_{\alpha.1}, L_{\alpha.2}, R_{\alpha.2}, L_{\alpha.2}, R_{\alpha.2}) \end{array}$$

since, for α there are three covers ξ_i, $1 \leq i \leq 3$

ξ_i	$\alpha.1$	$\alpha.2$	$\alpha.3$
	β_{i1}	β_{i2}	β_{i3}

The six decorations strings associated with the auxiliary tree sets β_i, $1 \leq i \leq 2$ have the form $\sigma^l_{\beta_{ik}} = aL_{\beta_{ik}.2}$ and $\sigma^r_{\beta_{ik}} = aR_{\beta_{ik}.2}$, for $1 \leq k \leq 3$. They yield the six clauses ($1 \leq i \leq 2$ and $1 \leq j \leq 3$)

$$\begin{array}{l} < \beta_i > (aL_{\beta_{i1}.2}, R_{\beta_{i1}.2}, aL_{\beta_{i2}.2}, R_{\beta_{i2}.2}, aL_{\beta_{i3}.2}, R_{\beta_{i3}.2}) \\ \quad \rightarrow < \beta_j > (L_{\beta_{i1}.2}, R_{\beta_{i1}.2}, L_{\beta_{i2}.2}, R_{\beta_{i2}.2}, L_{\beta_{i3}.2}, R_{\beta_{i3}.2}) \end{array}$$

since for β_1 and β_2 there are the six covers ξ_{ij}, $1 \leq i \leq 2$ and$1 \leq j \leq 3$

ξ_{ij}	$\beta_{i1}.2$	$\beta_{i2}.2$	$\beta_{i3}.2$
	β_{j1}	β_{j2}	β_{j3}

Finally the six trivial decoration strings of β_3, i.e., $\sigma^l_{\beta_{3k}} = \sigma^r_{\beta_{3k}} = \varepsilon$ for $1 \leq k \leq 3$ yield the single clause $< \beta_3 > (\varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon, \varepsilon) \rightarrow \varepsilon$, since there are no covers for β_3.[7]

If we apply to this particular case the general formula of [11], we get for the generated simple $2k$-PRCG G a worst case parse time complexity of $\mathcal{O}(|G|n^{2(k+v)})$ where v is the maximum number of adjunction nodes in an elementary tree set.

However, in the TAG case, since the decoration strings are well parenthesized, the generated simple 2-PRCG can be transformed into an equivalent 2-var form in which the dependance w.r.t. to the number of variables (i.e., twice the number of adjunction nodes in an elementary tree) can always be reduced to four, leading to the famous $\mathcal{O}(n^6)$. Thus, we can wonder whether a similar transformation cannot be performed in the MCTAG case. Unfortunately, the answer is no. This comes from the fact that, in the general case, the decoration strings of elementary tree sets are so completely interlaced that it is not possible to isolate one MCA site, without isolating the others. The search of an optimal solution (in which the arity k may change) is still an open problem.

3.3 Simple 2-PRCGs vs. TAGs

We have seen in Section 3.1 how any TAG can be transformed into a strongly equivalent simple 2-PRCG. We will show below, thanks to two examples, that there exist simple 2-PRCGs which cannot be transformed into equivalent TAGs. This shows that the set of all simple 2-PRCLs strictly contains that of TALs.

Example 4. Consider the language $L = \{a_1^n b_1^m a_2^n b_2^m \mid n, m > 0\}$ which may be seen as a transducted copy of $a_1^n b_1^m$ into $a_2^n b_2^m$. It is not difficult to see that L is a TAL. But if we add the constraint that both the a's and b's must be related in such a way that they are well parenthesized ((a_1, a_2) and (b_1, b_2) are parenthesized), one can show that this additional constraint cannot be fulfilled by a TAG. However the following simple 2-PRCG in 2-var-form defines L. But we can note that the second clause is not well-balanced.

$$
\begin{array}{ll}
S(XY) & \rightarrow S_0(X, Y) \\
S_0(L_A L_B, R_A R_B) & \rightarrow A(L_A, R_A)\ B(L_B, R_B) \\
A(a_1 L_A, R_A a_2) & \rightarrow A(L_A, R_A) \\
A(\varepsilon, \varepsilon) & \rightarrow \varepsilon \\
B(b_1 L_B, R_B b_2) & \rightarrow B(L_B, R_B) \\
B(\varepsilon, \varepsilon) & \rightarrow \varepsilon
\end{array}
$$

Example 5. It can be shown thanks to the pumping lemma for LCFRLs [12] that the language $L_2 = \{a^m b^m c^n d^n e^m f^m g^n h^n \mid m, n > 0\}$ is not a TAL. However it is defined by the following simple 2-RCL in 2-var form. Once again, we can see that the second clause is not well-balanced.

[7] We can note that this translation is not optimal in the sense that since β_1, β_2 and β_3 play a symetric role, the three predicates $< \beta_1 >$, $< \beta_2 >$ and $< \beta_3 >$ could be merged into a single predicate $< \beta_{123} >$.

$$
\begin{array}{ll}
S(XY) & \rightarrow [a^m b^m c^n d^n, e^m f^m g^n h^n](X, Y) \\
[a^m b^m c^n d^n, e^m f^m g^n h^n](XY, ZT) & \rightarrow [a^m b^m, e^m f^m](X, Z)\ [c^n d^n, g^n h^n](Y, T) \\
[a^m b^m, e^m f^m](aXb, eZf) & \rightarrow [a^m b^m, e^m f^m](X, Z) \\
[a^m b^m, e^m f^m](\varepsilon, \varepsilon) & \rightarrow \varepsilon \\
[c^n d^n, g^n h^n](cYd, gTh) & \rightarrow [c^n d^n, g^n h^n](Y, T) \\
[c^n d^n, g^n h^n](\varepsilon, \varepsilon) & \rightarrow \varepsilon
\end{array}
$$

In other words, simple 2-PRCGs are more powerful than TAGs, both from a weak and strong point of view.

If a simple 2-PRCG can be transformed into an equivalent simple 2-PRCG in 2-var form (not necessarily well-balanced), this PRCG can be parsed in $\mathcal{O}(n^6)$ time. However, there exist simple 2-PRCGs that cannot be transformed into equivalent 2-var form and thus cannot be parsed in $\mathcal{O}(n^6)$ (by the standard parsing algorithm).

Example 6. This is for example the case for a clause of the form

$$E(L_A L_B L_C L_D, R_C R_A R_D R_B) \rightarrow A(L_A, R_A)\ B(L_B, R_B)\ C(L_C, R_C)\ D(L_D, R_D)$$

for which there exist no non-empty substrings σ_1, σ_2, σ_3 and, σ_4, such that $\sigma_1\sigma_2 = L_A L_B L_C L_D$, $\sigma_3\sigma_4 = R_C R_D R_A R_B$ and σ_3 and σ_4 (or σ_4 and σ_3) contain all the closing patenthesizes of σ_1 and σ_2. Thus this clause, of parse time complexity $\mathcal{O}(n^{10})$, cannot be decomposed into more simple clauses of arity 2 with a number of variables less than 8.

4 Multi-Component TIGs

4.1 Definition

As mentioned in Section 2.1, a TIG is a restricted TAG where auxiliary trees must be either empty or left or right (auxiliary TIG tree for short) and adjunctions are disallowed on both root and foot nodes of auxiliary trees.

A Multi-Component TIG (MCTIG), is a MCTAG in which all trees in an auxiliary tree set are auxiliary TIG trees. Of course, if an adjunction node is on the spine of a left (resp. right) auxiliary tree, only a left (resp. right) auxiliary TIG tree or an empty tree can be adjoined at that node.

The maximum number k of auxiliary TIG trees in an auxiliary tree set is the *arity* of the MCTIG, we thus have a k-MCTIG.

4.2 TIGs are equivalent to Simple 1-PRCGs

Of course, a 1-MCTIG is a TIG. We will first see how it is possible to transform a TIG into an equivalent simple 1-PRCG. In fact this transformation is a simplification of the algorithm shown in Section 3.1 wich transforms a TAG into an equivalent PRCG: in all cases an elementary TIG tree gives birth to a

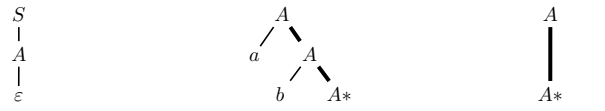
single decoration string during its top-down left-to-right traversal. The top-down traversal of the root node produces an empty symbol (since it is either an initial S-tree and there is no auxiliary S-trees or it is an auxiliary tree and adjunctions are not allowed at its root). The left-to-right traversal of leaf nodes labelled by a, $a \in T \cup \{\varepsilon\}$ produces the symbol a. The top-down (resp. bottom-up) traversal of an adjunction node η of label A produces the variable L_η (resp. R_η) if there exist left (resp. right) auxiliary A-trees or ε in the other cases.

Each non-empty elementary (initial or auxiliary) tree τ gives birth to a simple 1-PRCG clause c. Let σ be the decoration string of τ. If τ is an initial S-tree, the LHS of c is $S(\sigma)$, while if τ is a left (resp. right) auxiliary A-tree, the LHS of c is $[A]^l(\sigma)$ (resp. $[A]^r(\sigma)$).

If τ is a non-empty elementary tree with $|\tau|$ adjunction nodes, the RHS of c has the form $\phi^l_{\eta_1}\phi^r_{\eta_1} \ldots \phi^l_{\eta_{|\tau|}}\phi^r_{\eta_{|\tau|}}$. Let us denote by A_i the label of the node η_i. If L_{η_i} (resp. R_{η_i}) is a variable in σ we have $\phi^l_{\eta_i} = [A_i]^l(L_{A_i})$ (resp. $\phi^r_{\eta_i} = [A_i]^r(R_{A_i})$) or if L_{η_i} (resp. R_{η_i}) is not (a variable) in σ we have $\phi^l_{\eta_i} = \varepsilon$ (resp. $\phi^r_{\eta_i} = \varepsilon$).

If τ is an empty auxiliary A-tree, it gives birth to the pair of clauses $[A]^l(\varepsilon) \rightarrow \varepsilon$ and $[A]^r(\varepsilon) \rightarrow \varepsilon$.[8]

Example 7. We can find below the elementary trees of a TIG[9] which defines the language $\{a^n b^n \mid n \geq 0\}$. To each tree is associated its decoration string and the associated generated clauses. We can note that the absence of right auxiliary tree in that TIG implies that there are no R-variables and thus no $[]^r$-predicates.



$\sigma = L_A$	$\sigma = aL_Ab$	$\sigma = \varepsilon$
$S(L_A) \rightarrow [A]^l(L_A)$	$[A]^l(aL_Ab) \rightarrow [A]^l(L_A)$	$[A]^l(\varepsilon) \rightarrow \varepsilon$

In [13], Boullier has shown that any 1-RCG can always be transformed into an equivalent 1-RCG in 2-var form and can thus be parsed in $\mathcal{O}(n^3)$ time. This is an other way to show that TIGs can be parsed in $\mathcal{O}(n^3)$ time.

For space reasons, we will not show here how to perform the reverse conversion from a simple 1-PRCG to a TIG.[10] However, put together, these two conversion algorithms prove that simple 1-PRCGs are equivalent to TIGs and thus to 1-MCTIGs.

[8] Note that the first (resp. second) clause will never be used in any derivation if there is no left (resp. right) auxiliary A-trees. Of course, in that case, their generation may be skipped.

[9] Since there is no possible confusion, their nodes are addressed by their labels.

[10] The underlying idea is very simple: each clause is transformed into a (left) auxiliary tree: the root A_0 is the LHS predicate name; in A_0's (unique) argument, each sequence $u_i \in T*$ corresponds to leave nodes attached to the root, and each variable symbol $X_k \in V$ correspond to a subtree $\overset{X_k}{\underset{\epsilon}{|}}$ attached to the root.

Before considering the general case of the k-MCTIGs we study, in this Section, the relationships between TAGs, 2-MCTIGs and Simple 2-PRCGs. We first show that 2-MCTIGs and Simple 2-PRCGs are two equivalent formalisms.

4.3 2-MCTIGs and Simple 2-PRCGs are equivalent

To start, we show that a 2-MCTIG can be transformed into an equivalent simple 2-PRCG.

Let α be an initial tree set. By definition we have $|\alpha| = 1$ and τ is a S-tree if $\alpha = \{\tau\}$. The translation of α gives birth to a S-clause, where S is the unary start predicate. Let β be an auxiliary tree set. By definition we have $|\beta| \leq 2$. The translation of β gives birth to $[\beta]$-clauses, the predicate $[\beta]$ is unary if $|\beta| = 1$ or binay if $|\beta| = 2$.

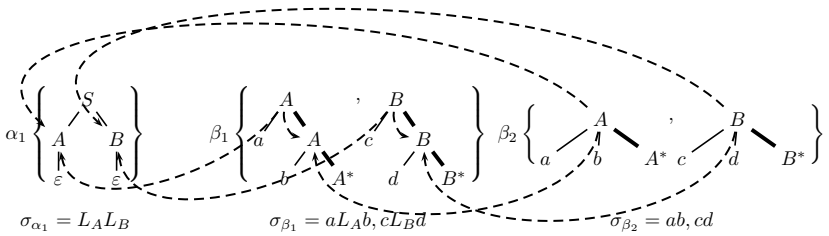
We will only consider there the more intricate case of a binary auxiliary tree set, leaving the reader to figure out how the other (simpler) cases can be processed. Let $\beta = \{\tau_1, \tau_2\}$ be a binary auxiliary tree set with its two decoration strings σ_{τ_1} and σ_{τ_2}, and with ξ one of its cover in $cover(\beta)$ (see Section 3.2). Let η_1 and η_2 be a pair of adjunction nodes in β and let τ'_1 and τ'_2 be the two auxiliary TIG trees of some auxiliary tree set β' selected by ξ (we have $\tau'_1 = \xi(\eta_1)$ and $\tau'_2 = \xi(\eta_2)$).

If τ'_1 is a left (resp. right) auxiliary tree, we erase the variable R_{η_1} (resp. L_{η_1}) from the (argument) string $(\sigma_{\tau_1}, \sigma_{\tau_2})$. Similarily, if τ'_2 is a left (resp. right) auxiliary tree, we erase the variable R_{η_2} (resp. L_{η_2}) from the string $(\sigma_{\tau_1}, \sigma_{\tau_2})$. Let K_{η_1} and K_{η_2} the variables which have thus been kept.

In the RHS of our clause we generate a binary predicate $[\beta']$ whose two arguments are either (K_{η_1}, K_{η_2}) or (K_{η_2}, K_{η_1}). It is (K_{η_1}, K_{η_2}) if τ'_1 is the first tree in the ordered tree set β' while it is (K_{η_2}, K_{η_1}) if τ'_1 is the second tree in β'.

Of course, we perform the same operation for all the other nodes of τ_1 and τ_2. When all the values of ξ are exhausted, we have built a RHS say ϕ_ξ and a new argument string $(\sigma'_{\tau_1}, \sigma'_{\tau_2})$. This yields the clause $[\beta](\sigma'_{\tau_1}, \sigma'_{\tau_2}) \rightarrow \phi_\xi$.

Example 8. We can find below the elementary trees of a 2-MCTIG which defines the language $\{a^n b^n c^n d^n \mid n > 0\}$. The various ξ's are represented by dashed arrows. Below each tree set we display its decoration string. Since this grammar is a left 2-MCTIG, there are no R-variables. Below each decoration string we show the generated clauses.



$S(L_A L_B) \rightarrow [\beta_1](L_A, L_B)$ $[\beta_1](aL_A b, cL_B d) \rightarrow [\beta_1](L_A, L_B)$ $[\beta_1](ab, cd) \rightarrow \varepsilon$
$S(L_A L_B) \rightarrow [\beta_2](L_A, L_B)$ $[\beta_1](aL_A b, cL_B d) \rightarrow [\beta_2](L_A, L_B)$ $[\beta_2](ab, cd) \rightarrow \varepsilon$

We can note that this translation is not optimal since the two predicates $[\beta_1]$ and $[\beta_2]$ play the same role, the can be merged and thus led to a simple 2-PRCG which only contains three clauses.

For space reasons we will not show here how to perform the reverse conversion from a simple 2-PRCG to a 2-MCTIG.[11] However, put together, these two conversion algorithms prove that simple 2-PRCGs are equivalent to 2-MCTIGs.

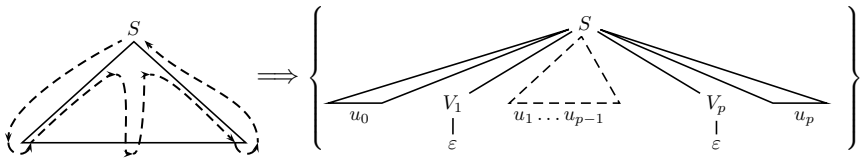
4.4 TAGs vs. 2-MCTIGs

In this section we show that any TAG can be transformed into an equivalent 2-MCTIG, but that the converse is not true.

Of course, to show that a TAG can be transformed into an equivalent 2-MCTIG, we can perform a TAG to PRCG transformation followed by a PRCG to MCTIG transformation. But below we propose a direct transformation, that easily extends to k-MCTAGs and $2k$-MCTIGs (see Section 4.6).

Let σ be a TAG tree decoration string. This decoration string is either associated to an initial TAG tree or associated to the left part or the right part of an auxiliary TAG tree. As usual, two variables L_η and R_η are associated to an adjunction node η and will occur in σ if that node is not on the spine. If η is on the spine of an auxiliary tree β, only L_η will occur in the left decoration string of β while only R_η will occur in the right decoration string of β. Now, we rewrite σ in changing each occurrence of L_η (resp. R_η) by A_l (resp. A_r) if A is the label of η.

This process will transform each initial TAG tree (a S-tree) into an initial tree set of cardinality 1 (this single tree is a S-tree) of the 2-MCTIG.



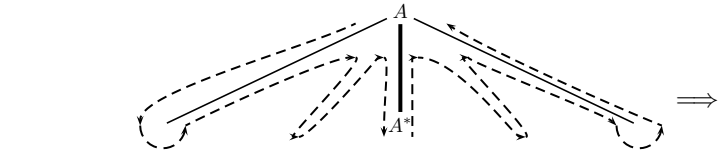
$\sigma = u_0 U_1 u_1 \dots U_p u_p$
$U_i = L_\eta$ or R_η

$V_i = A_l$ or A_r
where A is the label of η

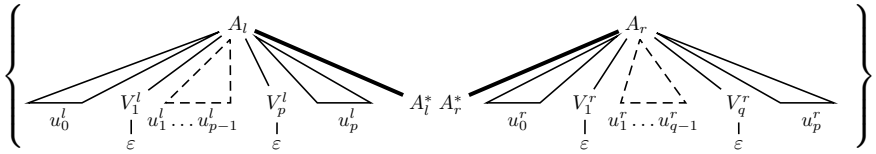
It will also transform each auxiliary TAG tree (say an A-tree) into an auxiliary ordered tree set of cardinality 2 (an A_l-tree and an A_r-tree) of the 2-MCTIG.

[11] Informally, it is a generalization of the transformation sketched in footnote 10. For a given clause, we first rename all variables (i.e., all RHS arguments) such that the first argument of predicate A_i is A_i^1 and its second argument is A_i^2. Then, if the LHS predicate A_0 is unary, it gives birth to an A_0^1-rooted left auxiliary tree in the same way as in footnote 10; if A_0 is binary, it gives birth to an ordered set of two trees rooted by A_0^1 (resp. A_0^2), and built similarily, but using only A_i^1s (resp. A_i^2s). This supposes that the 2-PRCG has been first transformed into a strongly equivalent grammar in a so-called *LR-form* (A_i^1s are only in A_0's first argument and A_i^2s only in A_0's second argument), which can be proven always possible.

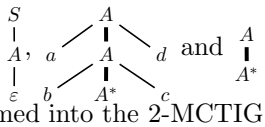
The A_l-tree (resp. A_r-tree) is built by the part of the auxiliary TAG tree to the left (resp. right) of its spine and by the spine itself.

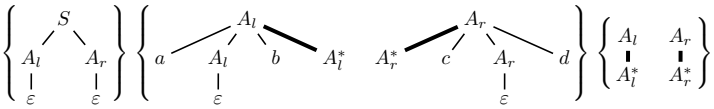


$\sigma_l = u_0^l U_1^l u_1^l \dots U_p^l u_p^l, \qquad \sigma_r = u_0^r U_1^r u_1^r \dots U_p^r u_p^r$
with U_i^l or $U_i^r = L_\eta$ or R_η



with V_i^l or $V_i{}^r = A_l$ or A_r s.t. A is the label of η

Example 9. For example the TAG  which defines the language $a^n b^n c^n d^n$, $n > 0$ is transformed into the 2-MCTIG



At this point, we know that any TAG can be transformed into a strongly equivalent simple 2-PRCG, and that simple 2-PRCGs and 2-MCTIGs are two equivalent formalisms. However, we have exhibited an example that shows that there are languages which can be defined by 2-MCTIG (or equivalently by simple 2-PRCGs) but which cannot be defined by TAGs.

4.5 $2k$-MCTIGs are equivalent to Simple $2k$-PRCGs

We have already studied in Section 4.3 the case $k = 1$ which has been distinguished because of the particular importance of TAGs (i.e., 1-MCTAGs). In the general case, the results are very similar.

First of all, h-MCTIGs and simple h-PRCGs are two equivalent formalisms, and the arity h remains the same. On the one hand, to show that a h-MCTIG can be transformed into an equivalent simple h-PRCG is only a trivial generalization of what we have seen in Section 4.3. On the other hand, to show that a simple h-PRCG can be transformed into an equivalent h-MCTIG is also only a trivial generalization of the algorithm we have evoked in Section 4.3.

4.6 k-MCTAGs vs. $2k$-MCTIGs

It is easy to show that a k-MCTAG can be transformed into an equivalent $2k$-MCTIG either indirectly by first using the transformation from MCTAG to simple PRCG of Section 3.2, followed by the transformation from a simple PRCG to a MCTIG mentioned in the same section. However, this transformation can be direct in generalizing the algorithm depicted in Section 4.4.

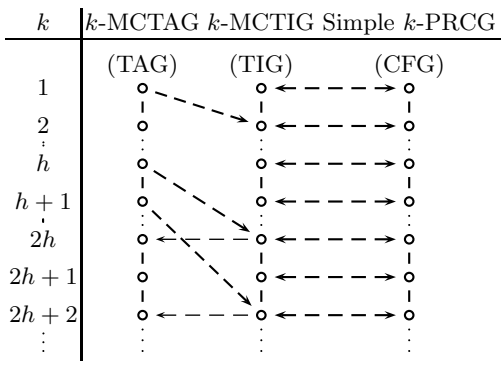
As for 2-MCTIG (and simple 2-PRCG), the converse is not true: $2k$-MCTIGs (and simple $2k$-PRCGs) are more powerful than k-MCTAGs.

However, MCTAGs, MCTIGs and simple PRCGs define the same class of languages since, by definition, a h-MCTIG may be seen as a h-MCTAG.

5 Conclusion: Hierarchies Comparison

Since k-MCTIGs and simple k-PRCGs are equivalent formalism, the hierarchy w.r.t. k of the class of languages defined by k-PRCGs, implies an identical hierarchy on the class of languages defined by k-MCTIGs. In particular let L_i be the counting language for i ($L_i = \{a_1^n \dots a_i^n \mid n \geq 0\}$), the languages L_{2h-1} and L_{2h} can be defined by a h-MCTIG while L_{2h+1}, L_{2h+2}, ... cannot. If we consider the class of MCTAGs, this figure slightly differs since the languages L_{4h-3}, L_{4h-2}, L_{4h-1} and L_{4h} can be defined by a h-MCTAG while L_{4h+1}, L_{4h+2}, ... cannot. This example shows that MCTAGs are more coarse-grained than MCTIGs in the sense that a single increment in the value of k delivers more (formal) power than the same increment in MCTIGs. In other words, for these three formalisms, in a $k+1$-class there exist languages that cannot be define by the corresponding k-class.

The following diagram summarizes, for each value of k and for each grammar class, in which other grammar class it can be transformed.



The direction of dashed arrows shows a language inclusion.

This diagram shows that for each value of k, k-MCTIGs and simple k-PRCGs are equivalent formalisms. But, with MCTAGs, the relation is not so simple. We know that a h-MCTAG can be transformed into an equivalent simple $2h$-PRCG (and hence into an equivalent $2h$-MCTIG), but we know that $2h$-PRCGs (or $2h$-MCTIG) are more powerful than h-MCTAGs. On the other hand we know that a $2h$-MCTIG is, by definition a $2h$-MCTAG.

We can wonder whether there exists a value l with $h < l \leq 2h$ such that the two formalisms l-MCTAGs and a $2h$-MCTIGs are equivalent? The answer is no. Assume that such a value l exists. In that case we know that the counting language L_{4l} can be defined by a l-MCTAG but L_{4l} can only be defined by a k-MCTIG in which $k \geq 2l$. This is in contradiction with the fact that $h < l$.

From an operational point of view, the equivalence results presented in this paper show that we can parse k-MCTIGs with a worse-case parsing complexity of $O(|G|n^{k+v})$, where G is the equivalent PRCG and v the maximum number of adjunction nodes in an elementary tree set.

References

1. Joshi, A.K. In: How much context-sensitivity is necessary for characterizing structural descriptions — Tree Adjoining Grammars. Cambridge University Press, New-York, NY (1985) D. Dowty, L. Karttunen, and A. Zwicky (eds.).
2. Weir, D.: Characterizing Mildly Context-Sensitive Grammar Formalisms. PhD thesis, University of Pennsylvania, Philadelphia, PA (1988)
3. Joshi, A.K.: An introduction to tree adjoining grammars. In Manaster-Ramer, A., ed.: Mathematics of Language, John Benjamins, Amsterdam (1987) 87–114
4. Vijay-Shanker, K., Weir, D., Joshi, A.K.: Characterizing structural descriptions produced by various grammatical formalisms. In: Proceedings of the 25th Meeting of the Association for Computational Linguistics (ACL'87), Stanford University, CA (1987) 104–111
5. Boullier, P.: On TAG parsing. Traitement Automatique des Langues (T.A.L.) **41**(3) (2000) 759–793
6. Boullier, P.: On Multicomponent TAG parsing. In: 6$^{\text{me}}$ confrence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'99), Cargse, Corse, France (July 1999) 321–326 See also *Research Report N ° 3668* at `http://www.inria.fr/RRRT/RR-3668.html`, INRIA-Rocquencourt, France, Apr. 1999, 39 pages.
7. Barthlemy, F., Boullier, P., Deschamp, P., ric de la Clergerie: Guided parsing of range concatenation languages. In: Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01), University of Toulouse, France (July 2001) 42–49
8. Schabes, Y., Waters, R.C.: Tree insertion grammar: A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. Computational Linguistics **21** (1994)
9. Joshi, A.K., Levy, L., Takahashi, M.: Tree adjunct grammars. Journal of Computer and System Sciences **10** (1975) 136–163
10. Boullier, P.: A generalization of mildly context-sensitive formalisms. In: Proceedings of TAG+4, University of Pennsylvania, Philadelphia, PA (August 1998) 17–20
11. Boullier, P.: Range Concatenation Grammars. In: New Developments in Parsing Technology. H. bunt, j. carroll, and g. satta edn. Volume 23 of Text, Speech and Language Technology. Kluwer Academic Publishers (2004) 269–289
12. Seki, H., Matsumura, T., Fujii, M., Kasami, T.: On multiple context-free grammars. Theoretical Computer Science **88** (1991) 191–229
13. Boullier, P.: A cubic time extension of context-free grammars. Grammars **3**(2/3) (2000) 111–131