On TAG and Multicomponent TAG Parsing

Pierre Boullier

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On TAG and Multicomponent TAG Parsing

Pierre Boullier

N ° 3668

Avril 1999

THÈME 3

ISSN 0249-6399

On TAG and Multicomponent TAG Parsing

Pierre Boullier*

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet Atoll

Rapport de recherche n ° 3668 — Avril 1999 — 39 pages

Abstract: The notion of mild context-sensitivity is an attempt to express the formal power needed to define the syntax of natural languages. However, all incarnations of mildly context-sensitive formalisms are not equivalent. On the one hand, near the bottom of the hierarchy, we find tree adjoining grammars and, on the other hand, near the top of the hierarchy, we find multicomponent tree adjoining grammars. This paper proposes a polynomial parse time method for these two tree rewriting formalisms. This method uses range concatenation grammars as a high-level intermediate definition formalism, and yields several algorithms. Range concatenation grammar is a syntactic formalism which is both powerful, in so far as it extends linear context-free rewriting systems, and efficient, in so far as its sentences can be parsed in polynomial time. We show that any unrestricted tree adjoining grammar can be transformed into an equivalent range concatenation grammar which can be parsed in $\mathcal{O}(n^6)$ time, and, moreover, if the input tree adjoining grammar has some restricted form, its parse time decreases to $\mathcal{O}(n^5)$. We generalize one of these algorithms in order to process multicomponent tree adjoining grammars. We show some upper bounds on their parse times, and we introduce a hierarchy of restricted forms which can be parsed more efficiently. Our approach aims at giving both a new insight into the multicomponent adjunction mechanism and at providing a practical implementation scheme.†

Key-words: range concatenation grammars, tree adjoining grammars, multicomponent tree adjoining grammars, mildly context-sensitive grammars, parsing time complexity.

(Résumé : tsvp)

* E-mail: Pierre.Boullier@inria.fr

† This report is an extended version of both [Boullier, 1999b] and [Boullier, 1999c]

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : 01 39 63 55 11 - International : +33 1 39 63 55 11
Télécopie : (33) 01 39 63 53 30 - International : +33 1 39 63 53 30

Sur l'analyse des TAG et des TAG ensemblistes

Résumé : La notion de formalismes faiblements contextuels correspond à une tentative pour exprimer le pouvoir formel dont on a besoin pour définir la syntaxe des langues naturelles. Cependant, toutes les incarnations de ces formalismes ne sont pas équivalentes. D'un côté, en bas de la hiérarchie, on trouve les grammaires d'arbres adjoints et, d'un autre côté, vers le haut de cette hiérarchie, on trouve les grammaires d'arbres adjoints ensemblistes. Ce rapport propose une méthode d'analyse pour ces deux formalismes de réécriture d'arbres. Cette méthode, qui se décline en plusieurs algorithmes, utilise les grammaires à concaténation d'intervalles comme formalisme de définition intermédiaire de haut niveau. La grammaire à concaténation d'intervalles est un formalisme syntaxique qui est à la fois puissant, puisqu'il étend les systèmes de réécriture non contextuels linéaires et efficace, puisque ses phrases peuvent être analysées en temps polynomial. Nous montrons que toute grammaire d'arbres adjoints, sans restrictions, peut être analysée en temps $\mathcal{O}(n^6)$ et que, de plus, si la grammaire d'arbres adjoints source a une certaine forme restreinte, le temps d'analyse diminue en $\mathcal{O}(n^5)$. On généralise l'un de ces algorithmes au traitement des grammaires d'arbres adjoints ensemblistes. On montre des bornes supérieures pour le temps d'analyse des grammaires d'arbres adjoints ensemblistes et l'on introduit une hiérarchie de formes restreintes qui bénéficient d'un temps d'analyse moindre. Le but de notre approche est double : proposer une nouvelle façon d'appréhender le méchanisme d'adjonction ensembliste et fournir un schéma d'implantation réaliste[‡].

Mots-clé : grammaires à concaténation d'intervalles, grammaires d'arbres adjoints, grammaires d'arbres adjoints ensemblistes, grammaires faiblement contextuelles, complexité en temps d'analyse.

[‡] Ce rapport est une version ´etendue de [Boullier, 1999b] et de [Boullier, 1999c]

1 Introduction

The notion of mild context-sensitivity is an attempt (see [Joshi, 1985] and [Weir, 1988]) to express the formal power needed to define the syntax of natural languages. However, all incarnations of mildly context-sensitive formalisms are not equivalent. On the one hand, near the bottom of the hierarchy, we find tree adjoining grammars (TAGs) [Vijay-Shanker, 1987] and some other weakly equivalent formalisms, linear indexed grammars (LIGs) [Gazdar, 1985], head grammars (HGs) [Pollard, 1984] and combinatory categorial grammars [Steedman, 1987]) (see [Vijay-Shanker and Weir, 1994a] for a proof of their equivalence). On the other hand, near the top of the hierarchy, we find multicomponent tree adjoining grammars (MC-TAGs) which are, in turn, equivalent to linear context-free rewriting systems (LCFRS) [Vijay-Shanker, Weir, and Joshi, 1987] and multiple context-free grammars (M-CFGs) [Seki, Matsumura, Fujii, and Kasami, 1991]. This paper proposes a polynomial parse time method for these two tree rewriting formalisms. This method uses range concatenation grammars (RCGs) as a high-level intermediate definition formalism, and yields several algorithms.

The notion of RCG is introduced in [Boullier, 1998a]; it is a syntactic formalism which is a variant of simple literal movement grammar (LMG), described in [Groenink, 1997], and which is also related to the framework of LFP developed by [Rounds, 1988]. In fact it may be considered to lie halfway between their respective *string* and *integer* versions; RCGs retain from the string version of LMGs or LFPs the notion of concatenation, applying it to ranges (couples of integers which denote occurrences of substrings in a source text) rather than strings, and from their integer version the ability to handle only (part of) the source text (this later feature being the key to tractability). This formalism, which extends context-free grammars (CFGs), aims at being a convincing challenger as a syntactic base for various tasks, especially in natural language processing. We have shown that the positive version of RCGs, as simple LMGs or integer indexing LFPs, exactly covers the class *PTIME* of languages recognizable in deterministic polynomial time. Since the composition operations of RCGs are not restricted to be linear and non-erasing, its languages (RCLs) are not semi-linear. Therefore, RCGs are *not* mildly context-sensitive and are more powerful than LCFRS,[1] while staying computationally tractable: its sentences can be parsed in polynomial time. However, this formalism shares with LCFRS the fact that its derivations are CF (i.e. the choice of the operation performed at each step only depends on the object to be derived from). As in the CF case, its derived trees can be packed into polynomial sized parse forests. For a CFG, the components of a parse forest are nodes labeled by couples (A, ρ) where A is a nonterminal symbol and ρ is a range, while for an RCG, the labels have the form $(A, \vec{\rho})$ where $\vec{\rho}$ is a vector (list) of ranges. Since this grammar class is closed both under intersection and complementation and since this closure properties are satisfied without changing the component grammars, RCGs can be considered as a modular syntactic formalism. If we add the fact that RCGs are both simple and understandable, and that they can be used as a support for annotations such as probabilities, logical terms, feature structures, they seem to be an adequate formalism in which the syntax of natural languages (NLs) can be *directly* defined. In this paper we rather want to explore an *indirect* usage of RCGs as an intermediate implementation formalism. This usage of RCGs has already been studied in [Boullier, 1998a] for syntactic formalisms such as TAGs,

[1] In [Boullier, 1999a], we argue that this extra power can be used in natural language processing.

LIGs or LCFRS. The TAG parsing case has been particularly studied in [Boullier, 1998b], but the algorithm which transforms a TAG into an equivalent RCG is only proposed for a restricted class of adjunction constraints, and, moreover, the $\mathcal{O}(n^6)$ parse time is only reached when the original TAG is in a special normal form.

The subject of TAG parsing inspired many researches but they all failed to beat the $\mathcal{O}(n^6)$ parse time wall.[2] Thus, current researches in this field, can be divided into two parts. A first one which tries to find parsing methods which can be efficiently implemented, and a second one, which tries to identify linguistically significant TAG subclasses whose theoretical parse time is smaller than $\mathcal{O}(n^6)$. In this paper, we try to reconcile these two approaches, and we first apply our method to TAG parsing. We show two new ways to parse TAGs with RCGs. Both methods accept unrestricted TAGs as inputs, however, the first one directly produces an equivalent RCG which is, in turn, transformed into another RCG, while the second one first dissects the elementary trees of the input TAG before applying a transformation into an object RCG. Both methods produce an equivalent RCG which can be parsed in $\mathcal{O}(n^6)$ time. Moreover, if the initial TAG is in the restricted form introduced in [Satta and Schuler, 1998], we show that the corresponding RCG has an $\mathcal{O}(n^5)$ parse time. Afterwards, we generalize the first of these algorithms in order to process MC-TAGs. This approach results in a polynomial upper bound for the parse time of unrestricted MC-TAGs. The degree of this polynomial can be improved if we restrict MC-TAGs to specific subclasses. Moreover, our approach gives a new insight into the multicomponent adjunction mechanism while providing a practical implementation scheme.

2 Range Concatenation Grammars

This section introduces the notion of RCG, more details can be found in [Boullier, 1998a].

A *positive range concatenation grammar* (PRCG) $G = (N, T, V, P, S)$ is a 5-tuple where N is a finite set of *predicate names*, T and V are finite, disjoint sets of *terminal symbols* and *variable symbols* respectively, $S \in N$ is the *start predicate name*, and P is a finite set of *clauses*

$$\psi_0 \rightarrow \psi_1 \dots \psi_m$$

where $m \geq 0$ and each of $\psi_0, \psi_1, \dots, \psi_m$ is a *predicate* of the form

$$A(\alpha_1, \dots, \alpha_p)$$

where $p \geq 1$ is its *arity*, $A \in N$ and each of $\alpha_i \in (T \cup V)^*$, $1 \leq i \leq p$, is an *argument*.

Each occurrence of a predicate in the RHS of a clause is a predicate *call*, it is a predicate *definition* if it occurs in its LHS. Clauses which define predicate A are called A-clauses. This definition assigns a fixed arity to each predicate name. The arity of the start predicate is one. The *arity* k of a grammar (we have a k-PRCG), is the maximum arity of its predicates.

Lower case letters such as $a, b, c, \dots$ will denote terminal symbols, while upper case letters such as $L, R, W, X, Y, \dots$ will denote elements of V.

[2] Asymptotically faster methods are known, but they all possess large hidden constants. There are even some arguments (see [Satta, 1994]) against the existence of faster algorithms with small constants.

The language defined by a PRCG is based on the notion of *range*. For a given input string $w = a_1 \dots a_n$ a range is a couple (i, j), $0 \leq i \leq j \leq n$ of integers which denotes the occurrence of some substring $a_{i+1} \dots a_j$ in w. The number $j - i$ is its *size*. We will use several equivalent denotations for ranges: an explicit dotted notation like $w_1 \bullet w_2 \bullet w_3$ or, if w_2 extends from positions $i + 1$ through j, a tuple notation $\langle i..j \rangle_w$, or $\langle i..j \rangle$ when w is understood or of no importance. For a range $\langle i..j \rangle$, i is its *lower bound* and j is its *upper bound*. If $i = j$, we have an *empty* range. Of course, only consecutive ranges can be concatenated into new ranges. In any PRCG, terminals, variables and arguments in a clause are supposed to be bound to ranges by a substitution mechanism. An *instantiated clause* is a clause in which variables and arguments are consistently (w.r.t. the concatenation operation) replaced by ranges; its components are *instantiated predicates*.

For example, $A(\langle g..h \rangle, \langle i..j \rangle, \langle k..l \rangle) \rightarrow B(\langle g{+}1..h \rangle, \langle i{+}1..j{-}1 \rangle, \langle k..l{-}1 \rangle)$ is an instantiation of the clause $A(aX, bYc, Zd) \rightarrow B(X, Y, Z)$ if the source text $a_1 \dots a_n$ is such that $a_{g+1} = a, a_{i+1} = b, a_j = c$ and $a_l = d$. In this case, the variables X, Y and Z are bound to $\langle g{+}1..h \rangle$, $\langle i{+}1..j{-}1 \rangle$ and $\langle k..l{-}1 \rangle$ respectively.

For a given PRCG and an input string w, a *derive* relation, denoted by $\underset{G,w}{\Rightarrow}$, is defined on strings of instantiated predicates. If an instantiated predicate is the LHS of some instantiated clause, it can be replaced by the RHS of that instantiated clause.

The *language* of a PRCG $G = (N, T, V, P, S)$ is the set

$$\mathcal{L}(G) \quad = \quad \{w \mid S(\bullet w \bullet) \overset{+}{\underset{G,w}{\Rightarrow}} \varepsilon\}$$

An input string $w = a_1 \dots a_n$ is a sentence iff the empty string (of instantiated predicates) can be derived from $S(\langle 0..n \rangle)$. Note that the order of predicate calls in the RHS of a clause is of no importance.[3]

The arguments of a given predicate may denote discontinuous or even overlapping ranges. Fundamentally, a predicate name A defines a notion (property, structure, dependency, ...) between its arguments whose ranges can be arbitrarily scattered over the source text. PRCGs are therefore well suited to describe long distance dependencies. Overlapping ranges arise as a consequence of the non-linearity of the formalism. For example, the same variable (denoting the same range) may occur in different arguments in the RHS of some clause, expressing different views (properties) of the same portion of the source text. However, in this paper, we do not need the full power of RCGs and we will restrict our attention to the simple subclass of PRCGs.

A clause is *simple*[4] if it is

non-combinatorial: each argument of its RHS predicates consists of a single variable, and

non-erasing and linear: each of its variables appears exactly one time in its LHS and one time in its RHS.

[3] In [Boullier, 1998a], we also define negative RCG (NRCG), which allows negative predicate calls. These negative calls define the complement language w.r.t. T^* of their positive counterpart. A *range concatenation grammar* (RCG) is either a PRCG or a NRCG.

[4] This is not Groenink's definition of simple.

A simple RCG is an RCG in which all its clauses are simple. If the maximum number of variables per argument is h, we have an RCG in h-var form.

As an example, the following simple 3-PRCG in 3-var form defines the three-copy language $\{www \mid w \in \{a,b\}^*\}$ which is not a CFL and is not even a TAL.

$$\begin{array}{lcl} S(XYZ) & \rightarrow & A(X,Y,Z) \\ A(aX,aY,aZ) & \rightarrow & A(X,Y,Z) \\ A(bX,bY,bZ) & \rightarrow & A(X,Y,Z) \\ A(\varepsilon,\varepsilon,\varepsilon) & \rightarrow & \varepsilon \end{array}$$

A parsing algorithm for RCGs has been presented in [Boullier, 1998a]. For an RCG G and an input string of length n, it produces a parse forest in time polynomial with n and linear with $|G|$. The degree of this polynomial is at most the number of free (independent) bounds in any clause. If we consider a simple k-RCG in h-var form, its number of free bounds is less or equal to $k(h+1)$. More precisely, this number is $\max_{c_j \in P} \sum_{i=1}^{k_j}(1+h_i)$ if c_j denotes the j^{th} clause in P, k_j is the arity of its LHS predicate and h_i is the number of variables occurring in its i^{th} argument. In other terms, the parse time is at worst $\mathcal{O}(|G|n^d)$ with $d = \max_{c_j \in P}(k_j + v_j)$ where v_j is the number of different variables in c_j.

3 Tree Adjoining Grammars

The most popular incarnation of mildly context-sensitive formalisms is certainly the TAG formalism. Of course, a considerable amount of effort has been and is still being applied to implement efficient TAG parsers. Most of them directly use the input TAG to implement their parsing algorithm. Among this class of direct implementations, we can remark that a majority is based upon variants of Earley's algorithm.[5] The use of another formalism to implement TAGs is not new either, and in particular, LIGs have been used several times. Though context-sensitive linguistic phenomena seem to be more naturally expressed within the TAG formalism, from a computational point of view, many authors think that LIGs play a central role. For example, quoted from [Schabes and Shieber, 1994] "...the LIG version of a TAG grammar can be used for recognition and parsing. Because the LIG formalism is based on augmented rewriting, the parsing algorithms can be much simpler to understand and easier to modify, and no loss of generality is incurred". In [Vijay-Shanker and Weir, 1994b] TAGs are transformed into equivalent LIGs. In [Vijay-Shanker and Weir, 1993] LIGs are used to express the derivations of a sentence in TAGs. In [Vijay-Shanker, Weir and Rambow, 1995] the approach used for D-Tree Grammars (DTGs) parsing is to translate a DTG into a Linear Prioritized Multiset Grammar which is similar to a LIG but uses multisets in place of stacks. However, LIGs and TAGs are equivalent formalisms see [Vijay-Shanker and Weir, 1994a]. In this paper, we proposed to use a much more powerful formalism to implement TAGs, and we will show that the theoretical worst-case complexity of the RCG-equivalent of a TAG is of the same order of complexity.[6]

[5] Of course we can found a lot of other basic techniques such as CYK, LR, LC, ...

[6] Of course, this encouraging result must be practically validated by an implementation in order to show the impact of the hidden constants and to compare the average case complexity with other existing TAG parsers.

A TAG is a tree rewriting formalism where trees are composed by means of the operations of tree adjunction and substitution. An introduction to TAG can be found in [Joshi, 1987]. Here, we assume that the reader is familiar with the formalism and, in this introduction, we mainly want to summarize our notations.

A TAG is a tuple $\mathcal{T} = (V_N, V_T, \mathcal{I}, \mathcal{A}, S)$ where V_N and V_T are finite disjoint sets of nonterminal and terminal symbols, $\mathcal{I}$ is the finite set of *initial trees* and $\mathcal{A}$ is the finite set of *auxiliary trees*. Trees in $\mathcal{I}$ are denoted by α's and trees in $\mathcal{A}$ by β's. The nonterminal S is the start symbol. We refer to the trees in $\mathcal{I} \cup \mathcal{A}$ as *elementary trees*. The set of nodes (addresses) in an elementary tree τ is denoted by $\mathcal{N}_\tau$ and the set of all nodes in the elementary trees is denoted by $\mathcal{N}_{\mathcal{I}\cup\mathcal{A}}$. For each node η, we define its label $lab(\eta)$ to be an element of $V_N \cup V_T \cup \{\varepsilon\}$. A node labeled by an element of V_N (resp. $V_T \cup \{\varepsilon\}$) is called a nonterminal (resp. terminal) node. If a node is labeled X, we have an X-node. A (sub)tree whose root is labeled X is an X-tree. In any elementary tree τ, its root node denoted r_τ and its inside nodes are nonterminal nodes. The leaves of elementary trees are nonterminal or terminal nodes. Each auxiliary tree β has exactly one distinguished leaf denoted f_β and called its *foot* node. In an auxiliary tree both the root and foot nodes bear the same label. The path from root to foot is called a *spine*.

A non leaf node or a foot node is called an *adjunction node*. For each adjunction node η, we define its *selective-adjoining constraints* by means of a function named adj; $adj(\eta)$ is the set of auxiliary trees that can be adjoined at η and we write $nil \in adj(\eta)$ if adjunction at η is optional.

Though substitution can easily be simulated by an adjunction operation, in order to be fully general, we define TAGs with substitutions. A nonterminal leaf which is not a foot node is called a *substitution node*. For each substitution node η, we define $sbst(\eta)$ as the set of *initial* trees that can be substituted at η, and we write $nil \in sbst(\eta)$ if substitution is optional.[7]

The tree language of a TAG is defined as the set of all *complete* trees that can be composed, starting from an initial S-tree, and in which no more mandatory adjunction or substitution can take place. Its string language is merely the set of terminal yields of its tree language.

4 Basic Transformation from TAG to Simple PRCG

This section describes the algorithm, extracted from [Boullier, 1998b], upon which, the new algorithms are based.

In this section, we will restrict our attention to TAG with the following properties: its initial trees are all labeled by the start symbol S, which is not used somewhere else, and adjunction is not allowed neither at the root nor at the foot of any auxiliary tree but is mandatory on inside nodes. Note that the first restriction implies that substitution is disallowed (there is no substitution node), and that the second restriction implies that for every internal adjunction node η, we have $adj(\eta) = \{\beta \mid lab(r_\beta) = lab(\eta)\}$.

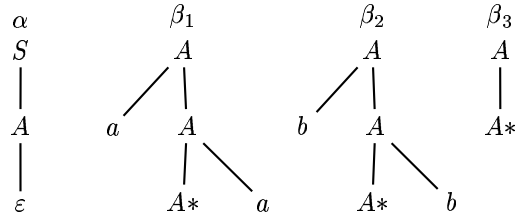
If we consider an auxiliary tree τ and the way it evolves until no more adjunction is possible, we realize that some properties of the final tree are already known on τ. The yield derived by the part of

[7] As usual in TAG theory, we assume that we have both $\beta \in adj(\eta), \beta \neq nil \Longrightarrow lab(\eta) = lab(r_\beta)$ and $\alpha \in sbst(\eta), \alpha \neq nil \Longrightarrow lab(\eta) = lab(r_\alpha)$.

τ to the left (resp. to the right) of its spine are contiguous and, the *left yield* (produced by the left part) lies to the left of the *right yield* in the 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_i and R_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. The root and foot of auxiliary trees have no decoration. Each terminal leaf has a single decoration which is its terminal symbol or ε. Afterwards, we collect into a string σ_τ the decorations gathered during a top-down left-to-right traversal of τ. 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 foot of τ has been traversed. With each elementary tree τ, we associate a simple PRCG clause constructed as follows:

- its LHS is the predicate $S(\sigma_\tau)$ if τ is an initial tree (S is the start predicate) or
- its LHS is the predicate $A(\sigma_\tau^l, \sigma_\tau^r)$ if τ is an auxiliary A-tree;
- its RHS is $\psi_1 \ldots \psi_m$ with $\psi_i = A_i(L_i, R_i)$ if $A_i = lab(\eta_i)$.

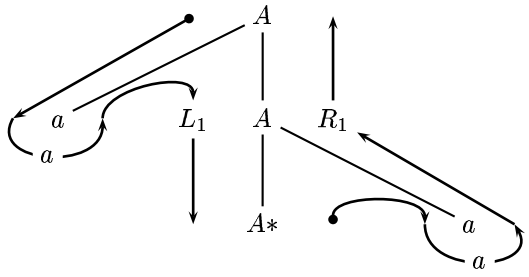
Example 1 *The following TAG*



where α is the initial tree and β_1, β_2 and β_3 are the auxiliary trees,[8] *defines the language $\{ww \mid w \in \{a,b\}^*\}$ which is translated into the strongly equivalent simple PRCG*

$$\begin{array}{lcl} S(L_1R_1) & \rightarrow & A(L_1,R_1) \\ A(aL_1,aR_1) & \rightarrow & A(L_1,R_1) \\ A(bL_1,bR_1) & \rightarrow & A(L_1,R_1) \\ A(\varepsilon,\varepsilon) & \rightarrow & \varepsilon \end{array}$$

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



[8] Each foot node is marked by an $*$.

We know that TAGs, LIGs and HGs are three weakly equivalent formalisms though they appear to have quite different external forms. Groenink has shown that HGs can be translated into equivalent simple LMGs. We have shown that transformation from TAGs to RCGs also exists. In [Boullier, 1998a] we have proposed a transformation from LIGs into equivalent RCGs. While the process involved to get an equivalent RCG for a TAG or an HG is rather straightforward, the equivalence proof for LIG is much more complex and relies upon our work described in [Boullier, 1996]. This is due to the fact that an RCG is a purely syntactic formalism in the sense that it only handles (part of) the source text, exclusive of any other symbol. Therefore the stack symbols of LIGs have no direct equivalent in RCGs and the translation process needs to understand what the structural properties induced by these stack symbols are. An interesting property of all these translations is that the power of RCGs comes for free. In particular, if the input TAG or LIG is in some normal form,[9] the corresponding RCG can be parsed in $\mathcal{O}(n^6)$ time at worst.

5 First Transformation from Unrestricted TAG to Simple PRCG

The algorithm in Section 4 is defined for a TAG with null adjunction constraint on its root and foot nodes (i.e. $adj(\eta) = \{nil\}$), an obligatory adjunction constraint on its internal adjunction nodes (i.e. $adj(\eta) = \{\beta_i \mid lab(r_{\beta_i}) = lab(\eta)\}$), and with no substitution site.

Here, we will extend this algorithm in such a way that it can handle unrestricted adjunction constraints. Afterwards, we will show that the generated simple PRCG has a particular form which allows for a transformation into an other equivalent simple PRCG which can, in all cases, be parsed in $\mathcal{O}(n^6)$ time at worst.

The algorithm which, starting from a TAG, produces an equivalent simple PRCG, can be summarized as follows.

Let $\mathcal{T} = (V_N, T, \mathcal{I}, \mathcal{A}, S)$ be an input TAG. For every elementary tree $\tau \in \mathcal{I} \cup \mathcal{A}$ we decorate each node $\eta \in \mathcal{N}_\tau$ with symbols

- if η is an adjunction node, it is decorated by two symbols L_η and R_η called LR*-variables*,
- if η is an substitution node, it is decorated by a single symbol S_η, its *substitution variable*, and
- if η is a terminal node, it has a single decoration which is $lab(\eta)$, its terminal label or ε.

The symbols L_η, R_η and S_η denote in fact variables which will be used in the clauses generated by the TAG to RCG transformation algorithm. The RCG variables L_η and R_η are left and right decorations which capture, as in the basic algorithm, respectively the *left yield* and the *right yield* of the trees that can be adjoined at η. Let η be a node of τ and x be the yield of the subtree rooted at η. After adjunction of a tree β into η (or any number of adjunctions within β), the yield of the subtree rooted at η becomes $x_l x x_r$, where x_l and x_r are respectively the left and right yield of β. The variables L_η and R_η span the occurrences (ranges) of x_l and x_r respectively. On the other hand, the RCG variable S_η captures the yields of the trees that can be substituted at η.

[9] Auxiliary trees in TAGs are such that they have at most two internal nodes where the adjunct operation can take place or the number of nonterminal symbols in the RHS of the LIG rules is at most two.

Afterwards, during a top-down left-to-right traversal of τ, we collect into a *decoration string* σ_τ the previous annotations. For an adjunction node, its L-variable is collected during its top-down traversal while its R-variable is collected during its bottom-up traversal. Substitution variables and terminal decorations are associated with leaves and are gathered during the traversal of this leaf node. If τ is an auxiliary tree, let σ_τ^l and σ_τ^r be the part of σ_τ gathered, during the previous traversal, respectively before and after the foot node of τ (i.e. we have $\sigma_\tau^l = L_{r_\tau} \dots L_{f_\tau}$ and $\sigma_\tau^r = R_{f_\tau} \dots R_{r_\tau}$).

Now, we are ready to describe the TAG to RCG transformation algorithm.

We will generate a simple PRCG $G = (N, T, V, P, S)$ which is equivalent to the initial TAG $\mathcal{T} = (V_N, T, \mathcal{I}, \mathcal{A}, S)$. We assume that the set N of its predicate names and the set V of its variables are implicitly defined by the clauses in P.

The generation process is divided into three phases, the initial phase, the main phase and the bond phase.

The *initial* phase simply connects the start symbol S with the initial S-trees. For every initial S-tree α, we generate

$$S(W) \quad \rightarrow \quad \alpha(W)$$

The *main* phase generates one clause for each elementary tree. Let τ be an elementary tree and let σ_τ be its decoration string. Of course, if τ is an auxiliary tree, σ_τ is cut into its left part and right part: $\sigma_\tau = \sigma_\tau^l \sigma_\tau^r$. With each such τ, we associate a simple clause, constructed as follows:

- its LHS is the predicate definition $\alpha(\sigma_\tau)$, if τ is the initial tree α;
- its LHS is the predicate definition $\beta(\sigma_\tau^l, \sigma_\tau^r)$, if τ is the auxiliary tree β;
- its RHS is $\psi_1 \dots \psi_i \dots \psi_m$, $1 \leq i \leq m, m = |\mathcal{N}_\tau|$ with
  - $\psi_i = [adj(\eta_i)](L_{\eta_i}, R_{\eta_i})$, if η_i is an adjunction node in $\mathcal{N}_\tau$,
  - $\psi_i = [sbst(\eta_i)](S_{\eta_i})$, if η_i is a substitution node in $\mathcal{N}_\tau$, and
  - $\psi_i = \varepsilon$, if η_i is a terminal node in $\mathcal{N}_\tau$.

The terms $[adj(\eta_i)]$ and $[sbst(\eta_i)]$ are predicate names which respectively denote the adjunction or substitution operations which can be performed at node η_i.

The *bond* phase connects the adjunction or substitution sites with the corresponding trees which are allowed by the adjunction or substitution constraints. For every nonterminal node $\eta \in \mathcal{N}_{\mathcal{I} \cup \mathcal{A}}$, we define the predicates $[adj(\eta_i)]$ and $[sbst(\eta_i)]$ by the following clauses

- If η is an adjunction node, for every $\tau \in adj(\eta)$ we produce the clause

  $$[adj(\eta)](L, R) \quad \rightarrow \quad \tau(L, R)$$

  if $\tau \in \mathcal{A}$ or

  $$[adj(\eta)](\varepsilon, \varepsilon) \quad \rightarrow \quad \varepsilon$$

  if $\tau = nil$.

- If η is a substitution node, for every $\tau \in sbst(\eta)$ we produce the clause

$$[sbst(\eta)](S) \quad \rightarrow \quad \tau(S)$$

if $\tau \in \mathcal{I}$ or

$$[sbst(\eta)](\varepsilon) \quad \rightarrow \quad \varepsilon$$

if $\tau = nil$.

Since these clauses are all simple and positive, and since predicates are all at most binaries, G is a simple 2-PRCG.[10]

Let us now examine the complexity of the parsing algorithm for G. From the general result on parsing with simple k-RCG in h-var form, we know that we have a polynomial time whose degree d is at worst $k(h+1)$. But, of course, our purpose is to express the result as a function of the initial TAG parameters. If we consider the more precise formula $d = \sum_{i=1}^{k}(1+h_i)$, since for G we trivially have $k = 2$, this gives $d = (1+h_1)+(1+h_2)$ where h_1 and h_2 are the number of variables of the first and second argument respectively. If we assume that p is the maximum number of adjunction nodes in the elementary trees, since each such node is decorated by two LR-variables, we have $h_1 + h_2 = 2p$. Thus, in the worst case, we get a $\mathcal{O}(n^{2p+2})$ parse time for unrestricted TAGs.

The idea is now to see whether G can be transformed into an equivalent RCG G' with a better parse time, and in particular whether we can reached the classical $\mathcal{O}(n^6)$ bound. The purpose of what follows is to transform each previously generated clause into a set of equivalent clauses in such a way that the number of their variables (and thus the number of free bounds) is as small as possible.

If we consider the decoration string σ_τ associated with any elementary tree τ, and if we remember that this string has gathered LR-variables during a top-down left-to-right traversal of τ, it is not difficult to figure out that σ_τ is a well parenthesized (Dyck) string where the couples of parentheses are the L and R-variables associated with its adjunction nodes and that its basic vocabulary is formed both by T, the set of terminal symbols, and by the set of its substitution variables.

Fundamentally, a Dyck language is recursively defined from initial strings in its basic vocabulary either by concatenation of two Dyck languages or by wrapping a Dyck language into a couple of parentheses. Conversely, each Dyck string can be recursively and unambiguously decomposed (parsed) either into concatenation of two Dyck strings or into a wrapped Dyck string. In our case, at each step of such decomposition of a Dyck (decoration) string σ, we can associate an RCG clause which exhibits either its decomposition into a wrapped prefix part σ_1 and a suffix part σ_2 or its unwrapping. However, this process is slightly complicated by the fact that σ, in the auxiliary tree case, is itself decomposed into two arguments, a left argument σ^l and a right argument σ^r. In what follows, we only process this case and we leave the reader to figure out how single argument predicates can be handled.

[10] In this paper, we will not address the correctness of the previous algorithm and we will assume that it generates a simple PRCG which is equivalent to the original TAG. The proof method relies upon the fact that the generated clauses are partitioned into three sets, the initial clauses (the S-clauses), the *main* clauses (the α or β-clauses) which are in bijection with the elementary trees, and the *bond* clauses (the $[adj(\eta)]$ or $[subst(\eta)]$-clauses) which provide a link between an adjunction or substitution node and the trees which are adjoined or substituted at that node. Any TAG derivation can be mapped onto a PRCG derivation, and conversely the generated PRCG can drive a parsing algorithm that recovers the TAG derivations of the input string.

Thus, for every auxiliary tree β whose decoration string is $\sigma_\beta = \sigma^l_\beta \sigma^r_\beta$, instead of a single clause of the form

$$\beta(\sigma^l_\tau, \sigma^r_\tau) \quad \rightarrow \quad \psi_1 \ldots \psi_m$$

we generate the following sequence of clauses.

First, we produce

$$\beta(L, R) \quad \rightarrow \quad [\sigma^l_\beta, \sigma^r_\beta](L, R)$$

in which the left and right decoration strings σ^l_β and σ^r_β become parts of the name of a new predicate $[\sigma^l_\beta, \sigma^r_\beta]$. The decoration string $\sigma^l_\beta \sigma^r_\beta$ is a Dyck string which is decomposed as previously suggested. Assume that at one step of this decomposition we have to define the predicate $[\sigma_1, \sigma_2]$.

If $\sigma_1 \sigma_2$ is in T^*, we produce

$$[\sigma_1, \sigma_2](\sigma_1, \sigma_2) \quad \rightarrow \quad \varepsilon$$

If $\sigma_1 \sigma_2 = u X \sigma'$ with $u \in T^+$ and X is a variable (either an LR-variable or a substitution variable) we produce, if U_1 and U_2 are two new variables

$$[\sigma_1, \sigma_2](uU_1, U_2) \quad \rightarrow \quad [X\sigma'_1, \sigma_2](U_1, U_2)$$

if $\sigma_1 = uX\sigma'_1$ or

$$[\sigma_1, \sigma_2](u_1, u_2U_2) \quad \rightarrow \quad [X\sigma'_2](U_2)$$

if $\sigma_1 = u_1$, $\sigma_2 = u_2 X \sigma'_2$ and $u = u_1 u_2$. Note that, in this later case, with the predicate call $[X\sigma'_2](U_2)$, we enter the *single argument processing part* of G'.

If the leading variable of $\sigma_1\sigma_2$ is a substitution variable say S_η (i.e. $\sigma_1\sigma_2 = S_\eta \sigma'_1 \sigma_2$), we produce

$$[S_\eta \sigma'_1, \sigma_2](S_\eta U_1, U_2) \quad \rightarrow \quad [sbst(\eta)](S_\eta)\ [\sigma'_1, \sigma_2](U_1, U_2)$$

If the leading variable of $\sigma_1\sigma_2$ is an L-variable say L_η, we have $\sigma_1\sigma_2 = L_\eta \sigma' R_\eta \sigma''$. The generated clauses depend on whether R_η occurs in σ_1 or in σ_2.

If $\sigma_1 = L_\eta \sigma' R_\eta \sigma'_1$, we produce

$$[L_\eta \sigma' R_\eta \sigma'_1, \sigma_2](U'_1 U''_1, U_2) \quad \rightarrow \quad [L_\eta \sigma' R_\eta](U'_1)\ [\sigma'_1, \sigma_2](U''_1, U_2)$$

If $\sigma_1 = L_\eta \sigma'_1$, $\sigma_2 = \sigma'_2 R_\eta \sigma''_2$, we produce

$$[L_\eta \sigma'_1, \sigma'_2 R_\eta \sigma''_2](U_1, U'_2 U''_2) \quad \rightarrow \quad [L_\eta \sigma'_1, \sigma'_2 R_\eta](U_1, U'_2)\ [\sigma''_2](U''_2)$$

And last, if $\sigma_1\sigma_2$ has the form $L_\eta \sigma'_1 \sigma'_2 R_\eta$, we generate

$$[L_\eta \sigma'_1, \sigma'_2 R_\eta](L_\eta U_1, U_2 R_\eta) \quad \rightarrow \quad [adj(\eta)](L_\eta, R_\eta)\ [\sigma'_1, \sigma'_2](U_1, U_2)$$

We have seen that each clause of G can be translated into a set of equivalent simple clauses either in 2-var form for binary predicates (or in 3-var form for unary predicates). The 2-var form gives a maximum of six free bounds[11] (and the 3-var form gives a maximum of four free bounds). Thus, by a grammar transformation at the RCG level, we have a method which parses the language defined by an unrestricted TAG in worst case time $\mathcal{O}(n^6)$. We can remark that, for presentation purpose, we have used a two phase process which involves an intermediate RCG. Of course, the previous algorithm can be easily changed into a one pass algorithm which directly generates an $\mathcal{O}(n^6)$ simple 2-PRCG, equivalent to the original TAG.

6 Second Transformation from Unrestricted TAG to Simple PRCG

The idea of the algorithm depicted here is to dissect the elementary trees of a TAG in a way such that each excised subtree directly generates an $\mathcal{O}(n^6)$ parsable clause at worst.

We will use for TAGs the notations and the vocabulary defined in Section 3, however, we need some supplementary definitions.

The same symbol α or β will be used to denote both any elementary tree and the root node of that elementary tree. If η is not a leaf node in some elementary tree, its i^{th} daughter, from left to right, is denoted $\eta.i$. A node on a spine is termed as *spinal* node. If the root of a (sub)tree is a spinal node, we have a *spinal* tree. Thus, a spinal tree is a subtree of an auxiliary tree rooted on its spine. A tree rooted at node η is denoted either by $\overset{\circ}{\underset{\lhd\rhd}{\eta}}$ if it is a spinal tree or by $\overset{\circ}{\underset{\Delta}{\eta}}$ if it is a non spinal tree.[12] The over hanging circle depicts its root node while the underlying triangles depict its subtrees. For a given node η, we define a *full open* tree as the list of its daughter trees. This full open tree is noted $\underset{\lhd\rhd}{\eta}$ or $\underset{\Delta}{\eta}$ according as η is a spinal or a non spinal node. An *open* tree is a list of consecutive daughter trees $\{\eta.i, \eta.i_{+1}, \ldots, \eta.j\}$. In the sequel we will handle two kinds of open trees $\underset{\lhd i}{\eta}$ and $\underset{i\rhd}{\eta}$ where $\underset{\lhd i}{\eta}$ denotes the left daughters whose rank is less or equal than i and $\underset{i\rhd}{\eta}$ denotes the right daughters whose rank is greater or equal than i. By contrast with the term open, a "normal" tree will be sometimes called a *close* tree.

For a given TAG $\mathcal{T} = (V_N, T, \mathcal{I}, \mathcal{A}, S)$, we will build an equivalent simple 2-PRCG $G = (N, T, V, P, S)$. The start symbol of $\mathcal{T}$ is the start predicate of G and their terminals are equal. Except for S, the predicates names of N are the previously defined close and open tree denotations. The *spinal tree* predicates are binary while the *non spinal tree* predicates are unary. The set of variables is $V = \{L, R, X, Y, Z\}$ where L and R (the LR-variables) are always associated with adjunction nodes and denote the left and right yields of the trees that can be adjoined at that node. If a tree β is adjoined at node η, the variable L (resp. R) will be bound to the range of the *left yield* (resp. *right yield*) derived by the part of β to the left (resp. right) of its spine. The other variables

[11] This maximum value is reached for an unwrapping operation when each of the two LR-variables occurs in different arguments. This case corresponds to an adjunction at a node which lays on a spine.

[12] Note that the α initial trees are also denoted by $\overset{\circ}{\underset{\Delta}{\alpha}}$, and the β auxiliary trees by $\overset{\circ}{\underset{\lhd\rhd}{\beta}}$.

X,Y and Z will span the yields of subtrees. And last, the set of clauses P is built, following the transformation rules listed below.

In a first phase, we initialize P with clauses defining the start predicate S, these clauses are issued from the initial trees whose root is labeled by the start symbol S. For each tree $\alpha \in \mathcal{I}$ s.t. $lab(\alpha) = S$, we add to P

$$S(X) \quad \rightarrow \quad \overset{\circ}{\underset{\Delta}{\alpha}}\,(X) \tag{1}$$

Afterwards, the elementary trees of $\mathcal{I} \cup \mathcal{A}$ are processed in turn. Each elementary tree is cut into smaller pieces, starting from the root. Intuitively, at each step, a (sub)tree is cut into two parts, its root and its full open tree. The type of processing for the root node depends whether adjunctions are allowed or not. While the processing of a full open tree consists of a partitioning into its constituent parts which are either (smaller) open trees or close trees. Of course, this processing differs whether spinal or non spinal open trees are considered. We iterate until the leaves are reached. If we reach a substitution node, we apply the possible substitutions.

We first consider the transformation of non spinal close trees rooted at node η.

If $\overset{\circ}{\underset{\Delta}{\eta}}$ is a non trivial non spinal close tree, we have

$$\Longrightarrow \quad \begin{cases} \beta \in adj(\eta), & \overset{\circ}{\underset{\Delta}{\eta}}\,(L\ X\ R) \quad \rightarrow \quad \overset{\circ}{\underset{\triangleleft\triangleright}{\beta}}\,(L,R)\ \underset{1\triangleright}{\eta}\,(X) & (a) \\ nil \in adj(\eta), & \overset{\circ}{\underset{\Delta}{\eta}}\,(X) \quad \rightarrow \quad \underset{1\triangleright}{\eta}\,(X) & (b) \end{cases} \tag{2}$$

If $\overset{\circ}{\underset{\Delta}{\eta}}$ is a leaf tree, depending of its label, we have

for an empty leaf (i.e. $\varepsilon = lab(\eta)$)

$$\begin{array}{c} \vdots \\ \eta\ \varepsilon \end{array} \quad \Longrightarrow \quad \overset{\circ}{\underset{\Delta}{\eta}}\,(\varepsilon) \quad \rightarrow \quad \varepsilon \tag{3}$$

for a terminal leaf a (i.e. $a = lab(\eta), a \in T$)

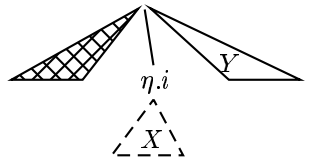
$$\begin{array}{c} \vdots \\ \eta\ a \end{array} \quad \Longrightarrow \quad \overset{\circ}{\underset{\Delta}{\eta}}\,(a) \quad \rightarrow \quad \varepsilon \tag{4}$$

for a substitution node (i.e. $A = lab(\eta), A \in V_N$)

$$\begin{array}{c} \vdots \\ \eta\ A \end{array} \Longrightarrow \left\{ \begin{array}{llll} \alpha \in sbst(\eta), & \overset{\circ}{\underset{\Delta}{\eta}}(X) & \rightarrow & \overset{\circ}{\underset{\Delta}{\alpha}}(X) \quad (a) \\ nil \in sbst(\eta), & \overset{\circ}{\underset{\Delta}{\eta}}(\varepsilon) & \rightarrow & \varepsilon \quad (b) \end{array} \right. \tag{5}$$

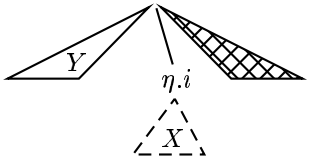
Now, we consider the transformation of non spinal open trees rooted at η. The components of such trees are processed from left to right in the case $\underset{i\triangleright}{\eta}$ or from right to left in the case $\underset{\triangleleft i}{\eta}$.

For an open tree of the form $\underset{i\triangleright}{\eta}$, we know that the left daughters of η whose rank is less than i must not be considered, thus such a tree is processed by extracting its i^{th} daughter and by iteratively processing $\underset{i+1\triangleright}{\eta}$, until completion. Of course, this completion is reached when, after the extraction of the i^{th} daughter, the resulting open tree is empty.



$$\Longrightarrow \left\{ \begin{array}{llll} Y \neq \emptyset, & \underset{i\triangleright}{\eta}(X\ Y) & \rightarrow & \overset{\circ}{\underset{\Delta}{\eta.i}}(X)\ \underset{i+1\triangleright}{\eta}(Y) \quad (a) \\ Y = \emptyset, & \underset{i\triangleright}{\eta}(X) & \rightarrow & \overset{\circ}{\underset{\Delta}{\eta.i}}(X) \quad (b) \end{array} \right. \tag{6}$$

For a right to left processing, we have



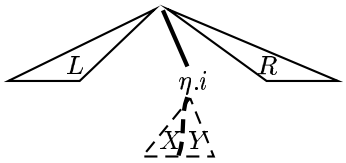
$$\Longrightarrow \left\{ \begin{array}{llll} i > 1, & \underset{\triangleleft i}{\eta}(Y\ X) & \rightarrow & \underset{\triangleleft i\text{-}1}{\eta}(Y)\ \overset{\circ}{\underset{\Delta}{\eta.i}}(X) \quad (a) \\ i = 1, & \underset{\triangleleft 1}{\eta}(X) & \rightarrow & \overset{\circ}{\underset{\Delta}{\eta.1}}(X) \quad (b) \end{array} \right. \tag{7}$$

Now, we consider the transformation of spinal trees rooted at η.

For each spinal close tree such as $\overset{\circ}{\underset{\triangleleft\triangleright}{\eta}}$, we must handle both a possible set of adjunctions at η, and the processing of the full open spinal subtree $\underset{\triangleleft\triangleright}{\eta}$. Thus, we have the transformation rule

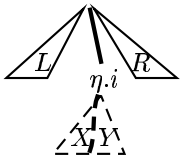
$$\Longrightarrow \left\{ \begin{array}{llll} \beta \in adj(\eta), & \overset{\circ}{\underset{\triangleleft\triangleright}{\eta}}(LX, YR) & \rightarrow & \overset{\circ}{\underset{\triangleleft\triangleright}{\beta}}(L, R)\ \underset{\triangleleft\triangleright}{\eta}(X, Y) \quad (a) \\ nil \in adj(\eta), & \overset{\circ}{\underset{\triangleleft\triangleright}{\eta}}(X, Y) & \rightarrow & \underset{\triangleleft\triangleright}{\eta}(X, Y) \quad (b) \end{array} \right. \tag{8}$$

For each spinal open tree $\underset{\triangleleft\triangleright}{\eta}$ rooted at η, we have, if $\eta.i$ is its spinal daughter node



$$\Longrightarrow \quad \underset{\lhd\rhd}{\eta}\,(LX,YR) \;\rightarrow\; \underset{\lhd i\text{-}1}{\eta}\,(L)\;\overset{\circ}{\underset{\lhd\rhd}{\eta.i}}\,(X,Y)\;\underset{i+1\rhd}{\eta}\,(R)$$

In that case, we note that the number of free bounds in the corresponding clause is six. We can remark that, on the one hand, the left and right open trees $\underset{\lhd i\text{-}1}{\eta}$ and $\underset{i+1\rhd}{\eta}$ are independent of each other and may thus be computed one after the other and, on the other hand, that any of them, or both, can be empty. Thus, equivalently, if η_r denotes an intermediate predicate name, we have the following transformation rules



$$\Longrightarrow \begin{cases} L\neq\emptyset, R\neq\emptyset, & \begin{cases} \underset{\lhd\rhd}{\eta}\,(LX,Y) \;\rightarrow\; \underset{\lhd i\text{-}1}{\eta}\,(L)\,\eta_r(X,Y) & (a)\\ \eta_r(X,YR) \;\rightarrow\; \overset{\circ}{\underset{\lhd\rhd}{\eta.i}}\,(X,Y)\;\underset{i+1\rhd}{\eta}\,(R) & (b)\end{cases}\\ L=\emptyset, R\neq\emptyset, & \underset{\lhd\rhd}{\eta}\,(X,YR) \;\rightarrow\; \overset{\circ}{\underset{\lhd\rhd}{\eta.i}}\,(X,Y)\;\underset{i+1\rhd}{\eta}\,(R) & (c)\\ L\neq\emptyset, R=\emptyset, & \underset{\lhd\rhd}{\eta}\,(LX,Y) \;\rightarrow\; \underset{\lhd i\text{-}1}{\eta}\,(L)\;\overset{\circ}{\underset{\lhd\rhd}{\eta.i}}\,(X,Y) & (d)\\ L=\emptyset, R=\emptyset, & \underset{\lhd\rhd}{\eta}\,(X,Y) \;\rightarrow\; \overset{\circ}{\underset{\lhd\rhd}{\eta.i}}\,(X,Y) & (e)\end{cases} \end{cases} \tag{9}$$

which generate clauses whose number of free bounds is now less or equal to five.

And last, for a foot node η, we have

$$\Longrightarrow \begin{cases} \beta\in adj(\eta), & \overset{\circ}{\underset{\lhd\rhd}{\eta}}\,(L,R) \;\rightarrow\; \overset{\circ}{\underset{\lhd\rhd}{\beta}}\,(L,R) & (a)\\ nil\in adj(\eta), & \overset{\circ}{\underset{\lhd\rhd}{\eta}}\,(\varepsilon,\varepsilon) \;\rightarrow\; \varepsilon & (b)\end{cases} \tag{10}$$

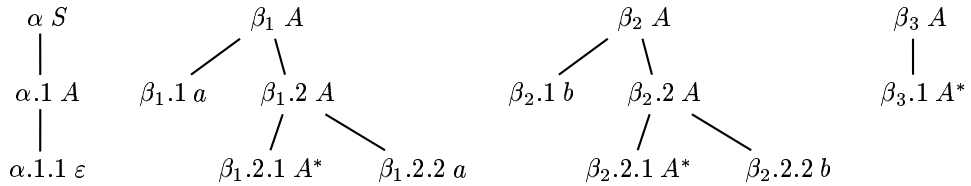
By inspection of all these generated clauses, we can verify that we have a simple 2-PRCG in 2-var form. The degree of the polynomial time complexity of this grammar is the maximum number of free bounds in each clause. We see that the maximum number of free bounds is six and is only reached one time in the clause resulting of transformation (8a). Transformations (9a–d) cost five (recall that they all arise as an optimization from a six free bound clause), while all the others contribute to a number less or equal to four. This transformation explicitly ranks by cost the operations used in TAG parsing. In particular, we confirm that the most costly operation is an adjunction at a spinal node.[13]

Thus, we have shown that any TAG can be translated into an equivalent simple 2-PRCG in 2-var form which can be parsed, at worst, in $\mathcal{O}(n^6)$ time.

[13] As for the previous algorithm, we leave aside the proof of the correctness of the above algorithm.

To account for the dependency of the input grammar size, we define $|\mathcal{T}| = \sum_{\eta \in \mathcal{N}_{\mathcal{I}} \cup \mathcal{A}} (1 + |adj(\eta)| + |sbst(\eta)|)$. We can easily see that the number of generated clauses is proportional to $|\mathcal{T}|$. Since the parse time of an RCG is linear in the size of its input grammar, we finally get an $\mathcal{O}(|\mathcal{T}|\, n^6)$ parse time for G.

Example 2 *Consider once again the TAG of Example 1, printed below with new node denotations:*



where α is the initial tree and β_1, β_2 and β_3 are the auxiliary trees. Each node is decorated by its label. We assume that we have the following constraints $adj(\beta_1) = adj(\beta_1.2.1) = adj(\beta_2) = adj(\beta_2.2.1) = adj(\beta_3) = adj(\beta_3.1) = \{nil\}$ and $adj(\alpha.1) = adj(\beta_1.2) = adj(\beta_2.2) = \{\beta_1, \beta_2, \beta_3\}$. This TAG is translated into the following equivalent set of clauses where each clause is annotated by the transformation rule used.

Initialization phase

$$S(X) \quad \rightarrow \quad \overset{\circ}{\underset{\Delta}{\alpha}}(X) \quad (1)$$

Transformation of α:

$$\begin{array}{lcll}
\overset{\circ}{\underset{\Delta}{\alpha}}(X) & \rightarrow & \underset{1\triangleright}{\alpha}(X) & (2b) \\
\underset{1\triangleright}{\alpha}(X) & \rightarrow & \overset{\circ}{\underset{\Delta}{\alpha.1}}(X) & (6b) \\
\overset{\circ}{\underset{\Delta}{\alpha.1}}(LXR) & \rightarrow & \overset{\circ}{\underset{\triangleleft\triangleright}{\beta_1}}(L,R)\ \underset{1\triangleright}{\alpha.1}(X) & (2a) \\
\overset{\circ}{\underset{\Delta}{\alpha.1}}(LXR) & \rightarrow & \overset{\circ}{\underset{\triangleleft\triangleright}{\beta_2}}(L,R)\ \underset{1\triangleright}{\alpha.1}(X) & (2a) \\
\overset{\circ}{\underset{\Delta}{\alpha.1}}(LXR) & \rightarrow & \overset{\circ}{\underset{\triangleleft\triangleright}{\beta_3}}(L,R)\ \underset{1\triangleright}{\alpha.1}(X) & (2a) \\
\underset{1\triangleright}{\alpha.1}(X) & \rightarrow & \overset{\circ}{\underset{\Delta}{\alpha.1.1}}(X) & (6b) \\
\overset{\circ}{\underset{\Delta}{\alpha.1.1}}(\varepsilon) & \rightarrow & \varepsilon & (3)
\end{array}$$

Transformation of β_1:

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_1}}\ (X,Y) \quad \rightarrow \quad \underset{\lhd\rhd}{\beta_1}\ (X,Y) \qquad (8b)$$

$$\underset{\lhd\rhd}{\beta_1}\ (L\ X,Y) \quad \rightarrow \quad \underset{\lhd 1}{\beta_1}\ (L)\ \ \overset{\circ}{\underset{\lhd\rhd}{\beta_1}}.2\ (X,Y) \qquad (9d)$$

$$\underset{\lhd 1}{\beta_1}\ (X) \quad \rightarrow \quad \overset{\circ}{\underset{\Delta}{\beta_1}}.1\ (X) \qquad (7b)$$

$$\overset{\circ}{\underset{\Delta}{\beta_1}}.1\ (a) \quad \rightarrow \quad \varepsilon \qquad (4)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_1}}.2\ (LX,YR) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_1}}\ (L,R)\ \ \underset{\lhd\rhd}{\beta_1}.2\ (X,Y) \qquad (8a)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_1}}.2\ (LX,YR) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_2}}\ (L,R)\ \ \underset{\lhd\rhd}{\beta_1}.2\ (X,Y) \qquad (8a)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_1}}.2\ (LX,YR) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_3}}\ (L,R)\ \ \underset{\lhd\rhd}{\beta_1}.2\ (X,Y) \qquad (8a)$$

$$\underset{\lhd\rhd}{\beta_1}.2\ (X,YR) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_1}}.2.1\ (X,Y)\ \ \underset{2\rhd}{\beta_1}.2\ (R) \qquad (9c)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_1}}.2.1\ (\varepsilon,\varepsilon) \quad \rightarrow \quad \varepsilon \qquad (10b)$$

$$\underset{2\rhd}{\beta_1}.2\ (X) \quad \rightarrow \quad \overset{\circ}{\underset{\Delta}{\beta_1}}.2.2\ (X) \qquad (6b)$$

$$\overset{\circ}{\underset{\Delta}{\beta_1}}.2.2\ (a) \quad \rightarrow \quad \varepsilon \qquad (4)$$

Transformation of β_2:

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_2}}\ (X,Y) \quad \rightarrow \quad \underset{\lhd\rhd}{\beta_2}\ (X,Y) \qquad (8b)$$

$$\underset{\lhd\rhd}{\beta_2}\ (L\ X,Y) \quad \rightarrow \quad \underset{\lhd 1}{\beta_2}\ (L)\ \ \overset{\circ}{\underset{\lhd\rhd}{\beta_2}}.2\ (X,Y) \qquad (9d)$$

$$\underset{\lhd 1}{\beta_2}\ (X) \quad \rightarrow \quad \overset{\circ}{\underset{\Delta}{\beta_2}}.1\ (X) \qquad (7b)$$

$$\overset{\circ}{\underset{\Delta}{\beta_2}}.1\ (b) \quad \rightarrow \quad \varepsilon \qquad (4)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_2}}.2\ (LX,YR) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_1}}\ (L,R)\ \ \underset{\lhd\rhd}{\beta_2}.2\ (X,Y) \qquad (8a)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_2}}.2\ (LX,YR) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_2}}\ (L,R)\ \ \underset{\lhd\rhd}{\beta_2}.2\ (X,Y) \qquad (8a)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_2}}.2\ (LX,YR) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_3}}\ (L,R)\ \ \underset{\lhd\rhd}{\beta_2}.2\ (X,Y) \qquad (8a)$$

$$\underset{\lhd\rhd}{\beta_2}.2\ (X,YR) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_2}}.1\ (X,Y)\ \ \underset{2\rhd}{\beta_2}.2\ (R) \qquad (9c)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_2}}.1\ (\varepsilon,\varepsilon) \quad \rightarrow \quad \varepsilon \qquad (10b)$$

$$\underset{2\rhd}{\beta_2}.2\ (X) \quad \rightarrow \quad \overset{\circ}{\underset{\Delta}{\beta_2}}.2\ (X) \qquad (6b)$$

$$\overset{\circ}{\underset{\Delta}{\beta_2}}.2.2\ (b) \quad \rightarrow \quad \varepsilon \qquad (4)$$

Transformation of β_3:

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_3}}\ (X,Y) \quad \rightarrow \quad \underset{\lhd\rhd}{\beta_3}\ (X,Y) \qquad (8b)$$

$$\underset{\lhd\rhd}{\beta_3}\ (X,Y) \quad \rightarrow \quad \overset{\circ}{\underset{\lhd\rhd}{\beta_3}}.1\ (X,Y) \qquad (9e)$$

$$\overset{\circ}{\underset{\lhd\rhd}{\beta_3}}.1\ (\varepsilon,\varepsilon) \quad \rightarrow \quad \varepsilon \qquad (10b)$$

7 Transformation from Restricted TAG to Simple PRCG

In [Satta and Schuler, 1998], the authors have identified a strict subclass of TAG which on the one hand can be parsed in $\mathcal{O}(n^5)$, and, on the other hand, as they argued, keeps enough generative power to capture the syntactic constructions of NLs that unrestricted TAGs can handle.[14]

[14]The claim of "linguistic relevance" of this subclass is neither discuss nor even advocate in this paper.

Let us call *restricted* TAG (RTAG), the subclass they defined. The purpose of this section is to show that any RTAG can be transformed into an equivalent simple PRCG whose parse time behavior is also $\mathcal{O}(n^5)$.

We will first introduce the RTAG subclass.

An auxiliary tree β is called a *right* (resp. *left*) tree if its leftmost (resp. rightmost) leaf is the foot node and if its spine contains only the root and the foot node. For a left tree β, its foot node is denoted by $\beta.s$ (it is its s^{th} daughter).

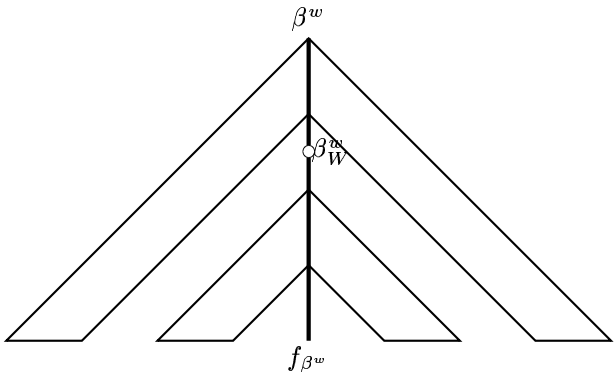
Right Tree Left Tree

An auxiliary tree which is neither left nor right is called a *wrapping* tree.

The restrictions imposed on RTAGs are twofold:[15]

1. on the spine of a wrapping tree, there is at most one node that allows adjunction of a wrapping tree. This node is called a *wrapping* node;
2. on the spine of each left (resp. right) tree, only left (resp. right) trees can be adjoined (the adjunction of wrapping trees is prohibited).

For a wrapping tree β^w, its wrapping node is denoted by β^w_W and its foot node is denoted by f_{β^w}.



Wrapping Tree

[15] In [Schabes and Waters, 1994], the authors defined tree insertion grammars (TIGs), a restricted form of TAGs in which only left and right trees (with a slightly more general definition), can be handled; wrapping trees are prohibited. Within these restrictions, they reach a cubic parse time for TIGs. It can be shown that TIGs can be transformed into an equivalent cubic time parsable 1-PRCG.

The above restrictions do not in any way constrain adjunction at nodes that are not on the spine, or adjunction of left or right trees on the spine of wrapping trees.

The algorithm of [Satta and Schuler, 1998] first performs a grammar transformation and afterwards applies a tabular evaluation working bottom-up on derivation trees. This algorithm is specified by means of inference rules. There are two ways to show that any RTAG can be transformed into an equivalent PRCG:

1. each inference rule is directly coded into equivalent PRCG clauses or
2. we directly transform the initial RTAG (without any grammar transformation) into an equivalent PRCG.

Below, we will investigate these two possibilities.

7.1 From Inference Rules to Simple PRCG

Here, we are not going to detail the transformation from the whole inference rule set to the corresponding set of clauses. We will only consider the most complicated inference rules and see how they can be translated. The most complicated step of the algorithm concern the adjunction of a wrapping tree at a wrapping node.

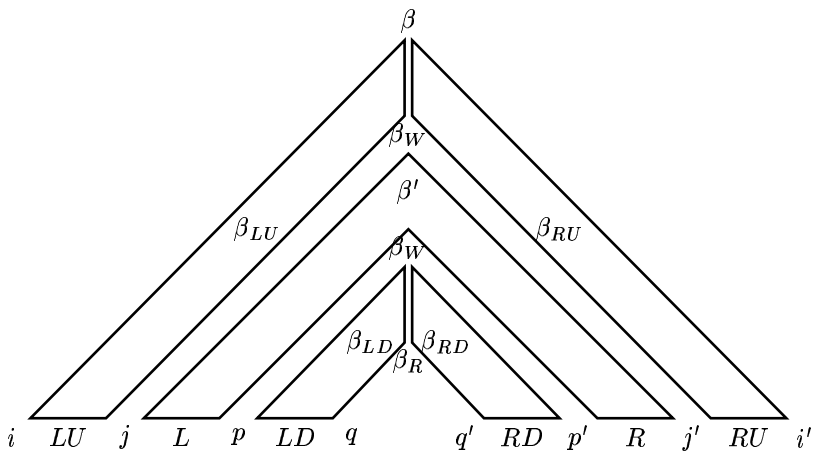


Table 1: Adjunction of a wrapping tree β' at a wrapping node β_W.

It is shown that in RTAGs, we can always *split* each wrapping tree β into four parts at some wrapping node, since this wrapping node is the only site on the spine that can host adjunction of a

wrapping tree. These four parts are named β_{LU}, β_{RU}, β_{LD} and β_{RD}, and respectively denote the parts of β which are left-up, right-up, left-down and right-down to the wrapping node. Adjunction of a wrapping tree β' at a wrapping node β_W can thus be simulated by wrapping β around β'. This simulation can be performed in four steps, executed one after the other. Each step composes the item resulting from the application of the previous step with an item representing one of the four parts of the wrapping tree β. For example if the tree β' can be adjoined at the wrapping node β_W of β, there is an item of the form $\langle \beta', j, p, p', j' \rangle$ where $\langle j..p \rangle$ is the range in the input string spanned by the left part of β' (i.e. the subtree of β' lying to the left of its spine) and $\langle p'..j' \rangle$ is the range in the input string spanned by the right part of β'. If moreover we assume that the left-down part of β has been recognized (i.e. there is an item of the form $\langle \beta_{LD}, p, q \rangle$ where $\langle p..q \rangle$, is the range spanning the left-down part of β, rooted at the wrapping node β_W),[16] we can state that the wrapping of β' by the left-down part of β has been performed, assuming that the upper bound of β_{LD} (i.e. q) is to the left of the lower bound of the right range of β' (i.e. p'). More formally, we have the following inference rule

$$\frac{\langle \beta_{LD}, p, q \rangle \ \langle \beta', j, p, p', j' \rangle}{\langle [\beta, LD], j, q, p', j' \rangle}, \beta' \in adj(W_\beta), q < p'$$

where $[\beta, LD]$ denotes the partial recognition (here the low-down corner) of the wrapping of β around the unique wrapping node β_W. If we assume a counterclockwise order, we will thus successively define $[\beta, RD]$, $[\beta, RU]$, and then β itself by

$$\frac{\langle \beta_{RD}, q', p' \rangle \ \langle [\beta, LD], j, q, p', j' \rangle}{\langle [\beta, RD], j, q, q', j' \rangle}, q < q'$$

$$\frac{\langle \beta_{RU}, j', i' \rangle \ \langle [\beta, RD], j, q, q', j' \rangle}{\langle [\beta, RU], j, q, q', i' \rangle}$$

$$\frac{\langle \beta_{LU}, i, j \rangle \ \langle [\beta, RU], j, q, q', i' \rangle}{\langle \beta, i, q, q', i' \rangle}$$

If we assume that the previous ranges are denoted by the following RCG variables

Range	$\langle i..j \rangle$	$\langle j..p \rangle$	$\langle p..q \rangle$	$\langle q'..p' \rangle$	$\langle p'..j' \rangle$	$\langle j'..i' \rangle$
Variable	LU	L	LD	RD	R	RU

the four inference rules can be directly translated into the four clauses

$$[\beta, LD](L\ LD, R) \rightarrow \beta_{LD}(LD)\ \ \beta'(L, R)$$

$$[\beta, RD](\underline{L\ LD}, RD\ R) \rightarrow \beta_{RD}(RD)\ \ [\beta, LD](\underline{L\ LD}, R)$$

$$[\beta, RU](\underline{L\ LD}, \underline{RD\ R}\ RU) \rightarrow \beta_{RU}(RU)\ \ [\beta, RD](\underline{L\ LD}, \underline{RD\ R})$$

$$\beta(LU\ \underline{L\ LD}, \underline{RD\ R\ RU}) \rightarrow \beta_{LU}(LU)\ \ [\beta, RU](\underline{L\ LD}, \underline{RD\ R\ RU})$$

where underlined sequences denote individual variables whose names indicate their (future) decomposition into their atomic constituent variables.

[16] Here, we ignore the three states B, M, and T indicating the progress of the recognition process.

Note that the first argument of binary predicates always denote a left subtree and the second argument a right subtree, thus any bound in the first argument is less or equal than the bounds in the second argument. This naturally fulfills the conditions we have mentioned on the application of the inference rules.

We easily verify that the maximum number of free bounds in the previous clauses is five.

The other inference rules of [Satta and Schuler, 1998], are simpler than those previously examined, and can be transformed into equivalent clauses whose number of free bounds is less than five.

Therefore, we get an equivalent 2-PRCG which, as the original RTAG, has an $\mathcal{O}(n^5)$ worse parse time behavior.

7.2 From RTAG to Simple PRCG

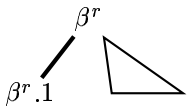
In this part we will study how the general transformation rules of Section 5 which transforms an unrestricted TAG onto an equivalent PRCG can be modified in order to produce for RTAGs an equivalent PRCG which can be parsed in worst case time $\mathcal{O}(n^5)$.

We assume that the auxiliary tree set $\mathcal{A}$ is partitioned into three sets $\mathcal{A}^l$, $\mathcal{A}^r$ and $\mathcal{A}^w$, which respectively contains left, right and wrapping trees denoted by β^l, β^r and β^w. The restrictions of the adj function to the codomains $\mathcal{A}^l$, $\mathcal{A}^r$ and $\mathcal{A}^w$ are denoted adj^l, adj^r and adj^w.

As in [Satta and Schuler, 1998],[17] without loss of generality and only for presentation facility, we assume that, in a wrapping tree, there is always a wrapping node and that it differs both from the root and the foot node.

First, let us consider the handling of left and right trees. For a left tree the adjunction constraints of its root and foot nodes impose that any number of compositions will result in a tree whose branches all lie to the left of its spine. Thus, we may consider that such an adjunction is a kind of substitution occurring at an inside node. Such a node spans a contiguous portion of the source text and can be represented by a single argument predicate. Of course, symmetric facts can be stated for right trees. The associated transformation rules are listed below.

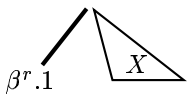
If β^r is the root node of a right tree, we add one of the clauses



$$\Longrightarrow \quad \begin{cases} \beta_i^r \in adj^r(\beta^r), & \underset{\triangleleft\triangleright}{\overset{\circ}{\beta}}{}^r\ (XR) \ \rightarrow\ \underset{\triangleleft\triangleright}{\beta^r}\ (X)\ \underset{\triangleleft\triangleright}{\overset{\circ}{\beta}}{}_i^r\ (R) \\ nil \in adj(\beta^r), & \underset{\triangleleft\triangleright}{\overset{\circ}{\beta}}{}^r\ (X) \ \rightarrow\ \underset{\triangleleft\triangleright}{\beta^r}\ (X) \end{cases}$$

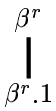
For each open tree $\underset{\triangleleft\triangleright}{\beta^r}$ where β^r is the root node of a right tree, we add one of the clauses

[17] However, we do not assume neither that elementary trees are in binary branching form, nor that no leaf is labeled by ε.



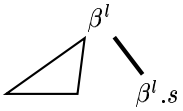
$$\Longrightarrow \quad \begin{cases} \beta_i^r \in adj^r(\beta^r.1), & \begin{cases} X \neq \emptyset, & \underset{\lhd\rhd}{\beta^r}\ (RX) & \rightarrow & \overset{\circ}{\underset{\lhd\rhd}{\beta_i^r}}\ (R)\ \underset{2\rhd}{\beta^r}\ (X) \\ X = \emptyset, & \underset{\lhd\rhd}{\beta^r}\ (R) & \rightarrow & \overset{\circ}{\underset{\lhd\rhd}{\beta_i^r}}\ (R) \end{cases} \\ nil \in adj(\beta^r.1), & \begin{cases} X \neq \emptyset, & \underset{\lhd\rhd}{\beta^r}\ (X) & \rightarrow & \underset{2\rhd}{\beta^r}\ (X) \\ X = \emptyset, & \underset{\lhd\rhd}{\beta^r}\ (\varepsilon) & \rightarrow & \varepsilon \end{cases} \end{cases}$$

Note that we decided to handle trivial auxiliary trees like



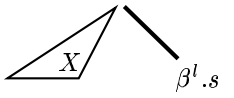
as right trees.

If β^l is the root node of a left tree, we add one of the clauses



$$\Longrightarrow \quad \begin{cases} \beta_i^l \in adj^l(\beta^l), & \overset{\circ}{\underset{\lhd\rhd}{\beta^l}}\ (LX) & \rightarrow & \overset{\circ}{\underset{\lhd\rhd}{\beta_i^l}}\ (L)\ \underset{\lhd\rhd}{\beta^l}\ (X) \\ nil \in adj(\beta^l), & \overset{\circ}{\underset{\lhd\rhd}{\beta^l}}\ (X) & \rightarrow & \underset{\lhd\rhd}{\beta^l}\ (X) \end{cases}$$

If β^l and $\beta^l.s$ are the root and foot nodes of the left tree β^l, for the open tree $\underset{\lhd\rhd}{\beta^l}$, we add one of the clauses



$$\Longrightarrow \quad \begin{cases} \beta_i^l \in adj^l(\beta^l.s), X \neq \emptyset, & \underset{\lhd\rhd}{\beta^l}\ (XL) & \rightarrow & \underset{\lhd s\text{-}1}{\beta^l}\ (X)\ \overset{\circ}{\underset{\lhd\rhd}{\beta_i^l}}\ (L) \\ nil \in adj(\beta^l.s), X \neq \emptyset, & \underset{\lhd\rhd}{\beta^l}\ (X) & \rightarrow & \underset{\lhd s\text{-}1}{\beta^l}\ (X) \end{cases}$$

For a wrapping tree β^w, the idea behind its processing is as follows. Starting from the root, we traverse its spine top-down, processing the nodes where the adjunction of left or right trees is allowed until we find the (only) wrapping node β_W^w. Then, we follow the spine bottom-up, starting from the foot node f_{β^w}, until we reach β_W^w, and then, only at that time, we process this wrapping node.

For each spinal node η of a wrapping tree β^w which strictly dominates the wrapping node β_W^w, the close spinal tree $\overset{\circ}{\underset{\lhd\rhd}{\eta}}$ is transformed by

$$\Longrightarrow \quad \begin{cases} \beta^l \in adj^l(\eta), & \overset{\circ}{\underset{\triangleleft\triangleright}{\eta}}\ (LX,Y) \ \rightarrow\ \overset{\circ}{\underset{\triangleleft\triangleright}{\beta^l}}\ (L)\ \underset{\triangleleft\triangleright}{\eta}\ (X,Y) \\ \beta^r \in adj^r(\eta), & \overset{\circ}{\underset{\triangleleft\triangleright}{\eta}}\ (X,YR) \ \rightarrow\ \underset{\triangleleft\triangleright}{\eta}\ (X,Y)\ \overset{\circ}{\underset{\triangleleft\triangleright}{\beta^r}}\ (R) \\ nil \in adj(\eta), & \overset{\circ}{\underset{\triangleleft\triangleright}{\eta}}\ (X,Y) \ \rightarrow\ \underset{\triangleleft\triangleright}{\eta}\ (X,Y) \end{cases}$$

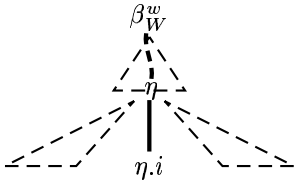
Each spinal open tree $\underset{\triangleleft\triangleright}{\eta}$, is processed by the unrestricted TAG transformation rules (9) of Section 5.

When, in some wrapping tree β^w, during the top-down traversal, we reach the wrapping node β^w_W, we trigger a bottom-up traversal, starting at the foot node f_{β^w}. Of course, during this traversal, the yields of the trees, rooted on the spine, and laying to the left (resp. right) of this spine are processed from right to left (resp. left to right). The symbols used to denote both the close and open (spinal) trees during this bottom-up traversal and their associated predicate names are respectively $\overset{\triangleleft\triangleright}{\underset{\circ}{\eta}}$ and $\overset{\triangleleft\triangleright}{\eta}$.

The bottom-up traversal is triggered by the clause

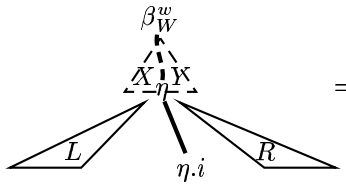
$$\overset{\circ}{\underset{\triangleleft\triangleright}{\beta^w_W}}\ (X,Y) \ \rightarrow\ \overset{\triangleleft\triangleright}{\underset{\circ}{f_{\beta^w}}}\ (X,Y)$$

For any spinal node $\eta.i$ (including the foot node), which is strictly dominated by the wrapping node β^w_W, the adjunction of left or right trees is described by



$$\Longrightarrow \quad \begin{cases} \beta^l \in adj^l(\eta.i), & \overset{\triangleleft\triangleright}{\underset{\circ}{\eta.i}}\ (XL,Y) \ \rightarrow\ \overset{\circ}{\underset{\triangleleft\triangleright}{\beta^l}}\ (L)\ \overset{\triangleleft\triangleright}{\eta.i}\ (X,Y) \\ \beta^r \in adj^r(\eta.i), & \overset{\triangleleft\triangleright}{\underset{\circ}{\eta.i}}\ (X,RY) \ \rightarrow\ \overset{\circ}{\underset{\triangleleft\triangleright}{\beta^r}}\ (R)\ \overset{\triangleleft\triangleright}{\eta.i}\ (X,Y) \\ nil \in adj(\eta.i), & \overset{\triangleleft\triangleright}{\underset{\circ}{\eta}}\ (X,Y) \ \rightarrow\ \overset{\triangleleft\triangleright}{\eta.i}\ (X,Y) \end{cases}$$

For any spinal node (including the foot node) $\eta.i$, which is strictly dominated by the wrapping node β^w_W, if $\eta.i_r$ denotes some intermediate predicate name, the processing of the associated bottom-up open spinal tree $\overset{\triangleleft\triangleright}{\eta.i}$, is described by



$$\Longrightarrow \begin{cases} L \neq \emptyset, R \neq \emptyset, & \begin{cases} \overset{\triangleleft\triangleright}{\eta.i}\ (XL, Y) & \rightarrow & \underset{\triangleleft i\text{-}1}{\eta}\ (L)\ \eta.i_r(X, Y) \\ \eta.i_r(X, RY) & \rightarrow & \underset{\circ}{\overset{\triangleleft\triangleright}{\eta}}\ (X, Y)\ \underset{i+1\triangleright}{\eta}\ (R) \end{cases} \\ L = \emptyset, R \neq \emptyset, & \overset{\triangleleft\triangleright}{\eta.i}\ (X, RY) & \rightarrow & \underset{\circ}{\overset{\triangleleft\triangleright}{\eta}}\ (X, Y)\ \underset{i+1\triangleright}{\eta}\ (R) \\ L \neq \emptyset, R = \emptyset, & \overset{\triangleleft\triangleright}{\eta}\ (XL, Y) & \rightarrow & \underset{\triangleleft i\text{-}1}{\eta}\ (L)\ \underset{\circ}{\overset{\triangleleft\triangleright}{\eta}}\ (X, Y) \\ L = \emptyset, R = \emptyset, & \overset{\triangleleft\triangleright}{\eta.i}\ (X, Y) & \rightarrow & \underset{\circ}{\overset{\triangleleft\triangleright}{\eta}}\ (X, Y) \end{cases}$$

And last, for each wrapping node β^w_W, we process the adjunction of wrapping trees

$$\Longrightarrow \begin{cases} \beta^w \in adj^w(\beta^w_W), & \underset{\circ}{\overset{\triangleleft\triangleright}{\beta^w_W}}\ (L, R) & \rightarrow & \underset{\triangleleft\triangleright}{\overset{\circ}{\beta^w}}\ (L, R) \\ nil \in adj(\beta^w_W), & \underset{\circ}{\overset{\triangleleft\triangleright}{\beta^w_W}}\ (\varepsilon, \varepsilon) & \rightarrow & \varepsilon \end{cases}$$

Finally, we just have to examine the adjunction operations at non spinal node. We get the analogous of the transformation rules (2) of Section 5.

If $\underset{\Delta}{\overset{\circ}{\eta}}$ denotes a non trivial non spinal tree rooted at η, we have

$$\Longrightarrow \begin{cases} \beta^l \in adj^l(\eta), & \underset{\Delta}{\overset{\circ}{\eta}}\ (L\ X) & \rightarrow & \underset{\triangleleft\triangleright}{\overset{\circ}{\beta^l}}\ (L)\ \underset{1\triangleright}{\eta}\ (X) \\ \beta^r \in adj^r(\eta), & \underset{\Delta}{\overset{\circ}{\eta}}\ (X\ R) & \rightarrow & \underset{\triangleleft\triangleright}{\overset{\circ}{\beta^r}}\ (R)\ \underset{1\triangleright}{\eta}\ (X) \\ \beta^w \in adj^w(\eta), & \underset{\Delta}{\overset{\circ}{\eta}}\ (L\ X\ R) & \rightarrow & \underset{\triangleleft\triangleright}{\overset{\circ}{\beta^w}}\ (L, R)\ \underset{1\triangleright}{\eta}\ (X) \\ nil \in adj(\eta), & \underset{\Delta}{\overset{\circ}{\eta}}\ (X) & \rightarrow & \underset{1\triangleright}{\eta}\ (X) \end{cases}$$

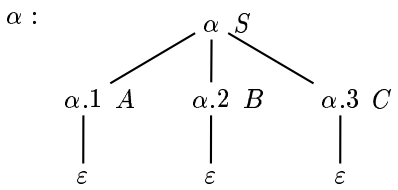
Thus, we have seen how any RTAG can be directly transformed into an equivalent simple 2-PRCG. Since in any clause produced by our transformation algorithm, there are at most five free bounds, the RCG parser works in worst case time $\mathcal{O}(n^5)$. Moreover, the general transformation rules for unrestricted TAGs and the previous rules for RTAG can be easily mixed up and lead to a unique transformation algorithm, working for any TAG and producing an equivalent simple PRCG. This PRCG has the following property, its worst case time is $\mathcal{O}(n^5)$ if the initial grammar is an RTAG or $\mathcal{O}(n^6)$ if the initial grammar is an unrestricted TAG.

8 Multicomponent TAGs

An extension of TAGs was introduced in [Joshi, Levy, and Takahashi, 1975] and later refined in [Joshi, 1987] where the adjunction operation involves a set of auxiliary trees instead of a single auxiliary tree. In a *multicomponent* TAG (MC-TAG), the elementary structures, both initial and auxiliary, instead of being two sets of single trees, as in TAGs, consist of two finite sets of finite tree sets. In MC-TAGs, 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 multicomponent adjunction (MCA) is defined as follows. All trees of an auxiliary tree set can be adjoined into distinct nodes (addresses) in a single elementary (initial or auxiliary) tree set.[18] Of course, if the cardinality of each tree set is one, an MC-TAG is a TAG. If the maximum cardinality of the initial tree sets is k, we have a k-MC-TAG. In [Weir, 1988], the author has shown that the languages defined by MC-TAGs are equal to the languages defined by LCFRS. The equivalence also holds with M-CFGs.[19]

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 cover the set-local interpretation of the term as it is more general and leads to equivalence to LCFRS. However, this report will not address the linguistic relevance of MC-TAGs (see for example [Kroch and Joshi, 1986], [Abeillé, 1994] and [Rambow and Lee, 1994]).

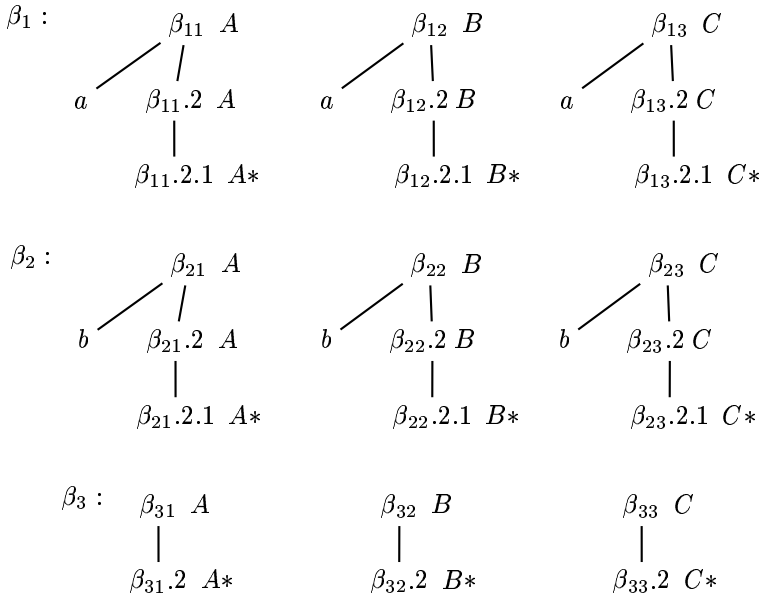
Example 3 (to be continued) *Let G be a set-local MC-TAG in which the set of initial tree sets is $\mathcal{I} = \{\{\alpha\}\}$ and the set of auxiliary tree sets is $\mathcal{A} = \{\beta_1, \beta_2, \beta_3\}$, where each tree set β_1, β_2 and β_3 contains three trees: $\beta_1 = \{\beta_{11}, \beta_{12}, \beta_{13}\}$, $\beta_2 = \{\beta_{21}, \beta_{22}, \beta_{23}\}$ and $\beta_3 = \{\beta_{31}, \beta_{32}, \beta_{33}\}$. The elementary trees of G are depicted below*[20]



[18] If the MCA operation is allowed into derived tree sets instead of elementary tree sets, we have a *nonlocal* MC-TAG. This version of MC-TAGs has not yet been studied in details, and it is not known whether nonlocal MC-TAGs are polynomially parsable (see [Becker, Joshi and Rambow, 1991]).

[19] This paper shows that any MC-TAG can be translated into an equivalent simple PRCG. In fact, we conjecture that the converse is true and that simple positive RCGs, set-local MC-TAGs, LCFRS, and M-CFGs are all weakly equivalent formalisms.

[20] 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 $\beta_{11}.2.1$.



If 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, 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 produces respectively three related a's or three related b's, while the MCA of (the trees in) β_3 terminates the process.

The adjunction and substitution constraints which apply to MC-TAGs are more complex to define than those which apply to TAGs. First, we choose to process substitutions as a particular case of MCAs and, second, we choose to identify (valid) MCAs by means of functions –the set of adjunction covers– defined below.

The reason why substitutions are treated as adjunctions is simply based upon the remark that the notion of substitution nodes disappears in MC-TAGs since adjunction of some tree in an elementary tree during an MCA can be hosted at some time by a leaf node (substitution) or, at some other time, by an internal node (adjunction).[21]

At each time, for an elementary tree set τ, we have to identify the nodes in $\mathcal{N}_\tau$ where MCAs can take place. This identification first results in a partition $\{\mathcal{N}_1, \ldots, \mathcal{N}_j, \ldots, \mathcal{N}_m\}$ of $\mathcal{N}_\tau$ in which each element $\mathcal{N}_j$ is the set of nodes at which a single MCA of some auxiliary tree set τ' can occur. Of course, if τ' is some auxiliary tree set β_i, we have $|\mathcal{N}_j| = |\beta_i|$ and if $\tau' = \emptyset$ (a nil adjunction), we have $|\mathcal{N}_j| = 1$. Second, for each node $\eta \in \mathcal{N}_j$, we must know which tree of β_i can be adjoined

[21] If adjunction is allowed at leaf nodes, note that the halting problem can be controlled by nil adjunction constraints at foot nodes. Equivalently, and without loss of generality, we can prohibit nonterminal leaf nodes (and thus substitutions) since, each such leaf node can be transformed into an inside node bearing a single empty daughter node.

at η. Thus we assume that for each $\mathcal{N}_j$, there is a bijective mapping ξ_j from $\mathcal{N}_j$ to β_i called *local adjunction cover*. We also define a function ξ, called *adjunction cover*, from $\mathcal{N}_\tau$ to $\cup_{\beta_i \in \mathcal{A}} \beta_i \cup \{nil\}$ s.t. $\forall j, 1 \leq j \leq m$, the restriction of ξ on $\mathcal{N}_j$ is the local adjunction cover ξ_j. Without loss of generality, we assume that all the MCA constraints of a given elementary tree set can be expressed by a finite set of adjunction covers.[22] Thus, associated with each elementary tree set τ, we assume that there is a finite set of adjunction covers. Each such adjunction cover ξ defines, on the one hand m local adjunction covers $\xi_1, \ldots, \xi_j, \ldots, \xi_m$, and, on the other hand, an m-partition $\Pi_\tau^\xi = \{\mathcal{N}_1, \ldots, \mathcal{N}_j, \ldots, \mathcal{N}_m\}$ of the adjunction nodes $\mathcal{N}_\tau$ of τ, s.t. each $\mathcal{N}_j$ is the definition domain $dom(\xi_j)$ of ξ_j, and s.t. for each node $\eta \in \mathcal{N}_\tau$, $\xi(\eta) \in \cup_{\beta_i \in \mathcal{A}} \beta_i \cup \{nil\}$ is the tree which is adjoined at η.

Example 3 (to be continued) *The various MCA constraints associated with the MC-TAG for the 3-copy language are expressed in terms of adjunction covers as follows*

- *There are three adjunction covers ξ_α^1, ξ_α^2, and ξ_α^3 for the initial tree set $\{\alpha\}$.*

$$\xi_\alpha^h = \{(\alpha, nil), (\alpha.1, \beta_{h1}), (\alpha.2, \beta_{h2}), (\alpha.3, \beta_{h3})\}, 1 \leq h \leq 3.$$

- *The corresponding partitions of $\mathcal{N}_\alpha$ are*

$$\Pi_\alpha^{\xi_\alpha^h} = \{\{\alpha\}, \{\alpha.1, \alpha.2, \alpha.3\}\}, 1 \leq h \leq 3.$$

- *For each auxiliary tree set β_1 and β_2, we have three adjunction covers.*
  If $1 \leq h \leq 3, 1 \leq i \leq 2$, these six functions are defined by

$$\begin{array}{rcl} \xi_{\beta_i}^h & = & \{(\beta_{i1}.2, \beta_{h1}), (\beta_{i2}.2, \beta_{h2}), (\beta_{i3}.2, \beta_{h3}), \\ & & (\beta_{i1}, nil), (\beta_{i2}, nil), (\beta_{i3}, nil), \\ & & (\beta_{i1}.2.1, nil), (\beta_{i2}.2.1, nil), (\beta_{i3}.2.1, nil)\} \end{array}$$

  and the corresponding partitions are

- $\Pi_{\beta_i}^{\xi_{\beta_i}^h} = \{\{\beta_{i1}\}, \{\beta_{i2}\}, \{\beta_{i3}\}, \{\beta_{i1}.2.1\}, \{\beta_{i2}.2.1\}, \{\beta_{i3}.2.1\}, \{\beta_{i1}.2, \beta_{i2}.2, \beta_{i3}.2\}\}$.
- *For β_3, we have a single adjunction cover*

$$\xi_{\beta_3} = \{(\beta_{31}, nil), (\beta_{32}, nil), (\beta_{33}, nil), (\beta_{31}.1, nil), (\beta_{32}.1, nil), (\beta_{33}.1, nil)\}$$

  and the corresponding partition is

- $\Pi_{\beta_3}^{\xi_{\beta_3}} = \{\{\beta_{31}\}, \{\beta_{32}\}, \{\beta_{33}\}, \{\beta_{31}.1\}, \{\beta_{32}.1\}, \{\beta_{33}.1\}\}$.

Now, we are ready to show that for any set-local k-MC-TAG there is an equivalent simple $2k$-PRCG.

[22] The way such a knowledge is acquired from a grammar, either by a direct specification or by some computation from more "local" adjunction constraints, such as the *adj* function, lies outside the scope of this paper.

9 Transformation from MC-TAG to Simple PRCG

The transformation of any MC-TAG into an equivalent simple PRCG is based upon a generalization of the basic transformation algorithm from TAG to simple PRCG proposed in Section 5.

Without loss of generality, we assume that initial tree sets are singletons whose root nodes are all labeled by the start symbol S. As for TAGs, each individual tree is decorated, and these annotations are gathered into decoration strings. The only difference is that, if we allow nonterminal leaf nodes, they are annotated, as the inside adjunction nodes, with two LR-variables (L_η and R_η for the node η instead of the single substitution variable S_η).

With each elementary tree set τ, we associate a single decoration string σ_τ, assuming some ordering on the trees of τ. For an initial tree set, which is a singleton, σ_τ is the decoration string of that single tree. For an auxiliary tree set β_i whose elements are the trees β_{i1}, ..., β_{ij}, ..., $\beta_{ip_i}, p_i = |\beta_i|$, its decoration string σ_{β_i} is the concatenation of the decoration strings $\sigma_{\beta_{ij}}$ of its component trees $\sigma_{\beta_i} = \sigma_{\beta_{i1}} \ldots \sigma_{\beta_{ij}} \ldots \sigma_{\beta_{ip_i}}$ where each substring $\sigma_{\beta_{ij}}$, as in TAGs, is cut in two parts $\sigma_{\beta_{ij}} = \sigma^l_{\beta_{ij}} \sigma^r_{\beta_{ij}}$, the left one has been collected before the foot node while the right one has been collected after the foot node. This means that $\sigma^l_{\beta_{ij}}$ has the form $L_{r_{\beta_{ij}}} \ldots L_{f_{\beta_{ij}}}$, while $\sigma^r_{\beta_{ij}}$ has the form $R_{f_{\beta_{ij}}} \ldots R_{r_{\beta_{ij}}}$ if the root and the foot nodes of β_{ij} are respectively denoted by $r_{\beta_{ij}}$ and $f_{\beta_{ij}}$.

For each elementary tree set τ, and for each adjunction cover ξ, we associate a unique clause $\psi_0 \rightarrow \psi_1 \ldots \psi_j \ldots \psi_m$, constructed as follows:

- If τ is an initial tree set α, we have $\psi_0 = S(\gamma^\xi_\alpha)$, where γ^ξ_α is a string built from the decoration string σ_α, in replacing each LR-variable L_η or R_η, such that $\xi(\eta) = nil$, by the empty string.

- If τ is an auxiliary tree set $\beta_i = \{\beta_{i1}, \ldots, \beta_{ij}, \ldots, \beta_{ip_i}\}$, we have $\psi_0 = \beta_i(\gamma^{\xi,l}_{\beta_{i1}}, \gamma^{\xi,r}_{\beta_{i1}}, \ldots, \gamma^{\xi,l}_{\beta_{ij}}, \gamma^{\xi,r}_{\beta_{ij}}, \ldots, \gamma^{\xi,l}_{\beta_{ip_i}}, \gamma^{\xi,r}_{\beta_{ip_i}})$ where $\gamma^{\xi,l}_{\beta_{ij}}$ and $\gamma^{\xi,r}_{\beta_{ij}}$ are respectively built from $\sigma^l_{\beta_{ij}}$ and $\sigma^r_{\beta_{ij}}$, the left and right decoration strings of the auxiliary tree β_{ij}, in replacing each LR-variable L_η or R_η, such that $\xi(\eta) = nil$, by the empty string. Note that the arity of the predicate name β_i is twice the cardinality of the auxiliary tree set β_i, since, as for TAGs, the description of each individual auxiliary tree takes two arguments.

- Its RHS is produced analogously, whether we consider initial or auxiliary tree sets. Let $\xi_1, \ldots, \xi_j, \ldots, \xi_m$ be the local adjunction covers of ξ and $\Pi^\xi_\tau = \{\mathcal{N}_1, \ldots, \mathcal{N}_j, \ldots, \mathcal{N}_m\}$, $\mathcal{N}_j = dom(\xi_j)$ be the corresponding partition of $\mathcal{N}_\tau$. For each local adjunction cover ξ_j whose codomain $codom(\xi_j)$ is not $\{nil\}$, if $\beta_l = codom(\xi_j) = \{\beta_{l1}, \ldots, \beta_{lh}, \ldots, \beta_{lk}\}$, we generate the predicate call $\psi_j = \beta_l(L_1, R_1, \ldots, L_h, R_h, \ldots, L_k, R_k)$ where $L_h = L_\eta$ and $R_h = R_\eta$ if $\beta_{lh} = \xi_j(\eta), \eta \in \mathcal{N}_j$. If $codom(\xi_j) = \{nil\}$, we have $\psi_j = \varepsilon$.

Here, we will assume the correctness of this algorithm.

However, we can easily check that this process only builds simple clauses and that the maximum arity of its predicates is $2k$, if we start from a k-MC-TAG.

Example 3 (to be continued) *The decoration string σ_α of the initial tree set $\{\alpha\}$ is*

- $\sigma_\alpha = L_\alpha L_{\alpha.1} R_{\alpha.1} L_{\alpha.2} R_{\alpha.2} L_{\alpha.3} R_{\alpha.3} R_\alpha$.

The decoration strings of the auxiliary tree sets β_i, $1 \leq i \leq 3$ *and their auxiliary trees* β_{ij}, $1 \leq j \leq 3$ *are*

- $\sigma_{\beta_i} = \sigma^l_{\beta_{i1}} \sigma^r_{\beta_{i1}} \sigma^l_{\beta_{i2}} \sigma^r_{\beta_{i2}} \sigma^l_{\beta_{i3}} \sigma^r_{\beta_{i3}}$, $1 \leq i \leq 3$ *such that*
  - $\sigma^l_{\beta_{ij}} = L_{\beta_{ij}} t_k L_{\beta_{ij}.2} L_{\beta_{ij}.2.1}$ *with* $1 \leq i \leq 2, 1 \leq j \leq 3, 1 \leq k \leq 2$ *and* $t_1 = a, t_2 = b$;
  - $\sigma^r_{\beta_{ij}} = R_{\beta_{ij}.2.1} R_{\beta_{ij}.2} R_{\beta_{ij}}$ *with* $1 \leq i \leq 2$ *and* $1 \leq j \leq 3$;
  - $\sigma^l_{\beta_{3j}} = L_{\beta_{3j}} L_{\beta_{3j}.1}$ *with* $1 \leq j \leq 3$;
  - $\sigma^r_{\beta_{3j}} = R_{\beta_{3j}.1} R_{\beta_{3j}}$ *with* $1 \leq j \leq 3$.

For the various adjunction covers, in erasing the LR*-variables associated with the* nil *constraints, we get*

- $\gamma^{\xi^h_\alpha}_\alpha = L_{\alpha.1} R_{\alpha.1} L_{\alpha.2} R_{\alpha.2} L_{\alpha.3} R_{\alpha.3}$ *with* $1 \leq h \leq 3$;
- $\gamma^{\xi^h_{\beta_i}, l}_{\beta_{ij}} = t_k L_{\beta_{ij}.2}$ *with* $1 \leq i \leq 2, 1 \leq j \leq 3, 1 \leq h \leq 3, 1 \leq k \leq 2, t_1 = a, t_2 = b$;
- $\gamma^{\xi^h_{\beta_i}, r}_{\beta_{ij}} = R_{\beta_{ij}.2}$ *with* $1 \leq i \leq 2, 1 \leq j \leq 3, 1 \leq h \leq 3$;
- $\gamma^{\xi_{\beta_3}, l}_{\beta_{3j}} = \gamma^{(\xi_{\beta_3}, r)}_{\beta_{3j}} = \varepsilon$ *with* $1 \leq j \leq 3$.

Afterwards, applying the previous transformation rules, we get the clauses

$$
\begin{array}{lcl}
S(L_{\alpha.1} R_{\alpha.1} L_{\alpha.2} R_{\alpha.2} L_{\alpha.3} R_{\alpha.3}) & \rightarrow & \beta_1(L_{\alpha.1},\ R_{\alpha.1},\ L_{\alpha.2},\ R_{\alpha.2},\ L_{\alpha.3},\ R_{\alpha.3}) \\
S(L_{\alpha.1} R_{\alpha.1} L_{\alpha.2} R_{\alpha.2} L_{\alpha.3} R_{\alpha.3}) & \rightarrow & \beta_2(L_{\alpha.1},\ R_{\alpha.1},\ L_{\alpha.2},\ R_{\alpha.2},\ L_{\alpha.3},\ R_{\alpha.3}) \\
S(L_{\alpha.1} R_{\alpha.1} L_{\alpha.2} R_{\alpha.2} L_{\alpha.3} R_{\alpha.3}) & \rightarrow & \beta_3(L_{\alpha.1},\ R_{\alpha.1},\ L_{\alpha.2},\ R_{\alpha.2},\ L_{\alpha.3},\ R_{\alpha.3}) \\
\beta_1(aL_{\beta_{11}.2},\ R_{\beta_{11}.2},\ aL_{\beta_{12}.2},\ R_{\beta_{12}.2},\ aL_{\beta_{13}.2},\ R_{\beta_{13}.2}) & & \\
\qquad \rightarrow \beta_1(L_{\beta_{11}.2},\ R_{\beta_{11}.2},\ L_{\beta_{12}.2},\ R_{\beta_{12}.2},\ L_{\beta_{13}.2},\ R_{\beta_{13}.2}) & & \\
\beta_1(aL_{\beta_{11}.2},\ R_{\beta_{11}.2},\ aL_{\beta_{12}.2},\ R_{\beta_{12}.2},\ aL_{\beta_{13}.2},\ R_{\beta_{13}.2}) & & \\
\qquad \rightarrow \beta_2(L_{\beta_{11}.2},\ R_{\beta_{11}.2},\ L_{\beta_{12}.2},\ R_{\beta_{12}.2},\ L_{\beta_{13}.2},\ R_{\beta_{13}.2}) & & \\
\beta_1(aL_{\beta_{11}.2},\ R_{\beta_{11}.2},\ aL_{\beta_{12}.2},\ R_{\beta_{12}.2},\ aL_{\beta_{13}.2},\ R_{\beta_{13}.2}) & & \\
\qquad \rightarrow \beta_3(L_{\beta_{11}.2},\ R_{\beta_{11}.2},\ L_{\beta_{12}.2},\ R_{\beta_{12}.2},\ L_{\beta_{13}.2},\ R_{\beta_{13}.2}) & & \\
\beta_2(bL_{\beta_{21}.2},\ R_{\beta_{21}.2},\ bL_{\beta_{22}.2},\ R_{\beta_{22}.2},\ bL_{\beta_{23}.2},\ R_{\beta_{23}.2}) & & \\
\qquad \rightarrow \beta_1(L_{\beta_{21}.2},\ R_{\beta_{21}.2},\ L_{\beta_{22}.2},\ R_{\beta_{22}.2},\ L_{\beta_{23}.2},\ R_{\beta_{23}.2}) & & \\
\beta_2(bL_{\beta_{21}.2},\ R_{\beta_{21}.2},\ bL_{\beta_{22}.2},\ R_{\beta_{22}.2},\ bL_{\beta_{23}.2},\ R_{\beta_{23}.2}) & & \\
\qquad \rightarrow \beta_2(L_{\beta_{21}.2},\ R_{\beta_{21}.2},\ L_{\beta_{22}.2},\ R_{\beta_{22}.2},\ L_{\beta_{23}.2},\ R_{\beta_{23}.2}) & & \\
\beta_2(bL_{\beta_{21}.2},\ R_{\beta_{21}.2},\ bL_{\beta_{22}.2},\ R_{\beta_{22}.2},\ bL_{\beta_{23}.2},\ R_{\beta_{23}.2}) & & \\
\qquad \rightarrow \beta_3(L_{\beta_{21}.2},\ R_{\beta_{21}.2},\ L_{\beta_{22}.2},\ R_{\beta_{22}.2},\ L_{\beta_{23}.2},\ R_{\beta_{23}.2}) & & \\
\beta_3(\varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon) & \rightarrow & \varepsilon
\end{array}
$$

Now we will address the parse time (of the RCG version) of k-MC-TAGs.

If we apply, to our case, the general formula which gives the degree of the polynomial parse time for a simple RCG i.e. $d = \max_{c_j \in P} \sum_{i=1}^{k_j}(1 + h_i)$, we get $d = \max_{c_j \in P}(k_j + v_j)$ if c_j is the j^{th} clause, v_j is the number of RCG variables within c_j, and k_j is the arity of its LHS predicate. In a k-MC-TAG, we have $k_j \leq 2k$ and if v is the maximum number of nonterminal nodes hosting a non nil adjunction constraint, we have $v_j \leq 2v$. Thus, any k-MC-TAG can be parsed at worst in $\mathcal{O}(n^{2(k+v)})$ time.

10 MC-TAG parsing optimization

For TAGs, we have found two generation techniques which both result in the famous $\mathcal{O}(n^6)$ parse time. We can remark that this parse time does not depend neither on the form of the elementary trees nor on the number of adjunction nodes in a tree. The transformation in Section 5 operates at the RCG level and produces an equivalent simple PRCG in 2-var form. This transformation directly relies on the fact that the decoration strings of its elementary trees are elements of a Dyck language where the parentheses are couples of variables such as (L_η, R_η). Of course, the question whether the generalization of such a technique applies to MC-TAGs, comes immediately at mind.

Unfortunately, the answer is no. For an MC-TAG, we consider the set of nodes $\mathcal{N}_\tau$ of an elementary tree set τ, a partition $\Pi_\tau = \{\mathcal{N}_1, \ldots, \mathcal{N}_i, \ldots, \mathcal{N}_m\}$ of $\mathcal{N}_\tau$ s.t. an MCA can take place at each $\mathcal{N}_i$, and the tuples $\pi_i = (L_{\eta_1}, R_{\eta_1}, \ldots, L_{\eta_j}, R_{\eta_j}, \ldots L_{\eta_{|\mathcal{N}_\tau|}}, R_{\eta_{|\mathcal{N}_\tau|}})$ containing the LR-variables L_{η_j}, R_{η_j}, $\eta_j \in \mathcal{N}_i$ in some order. It is not difficult to see that the decoration string σ_τ associated with τ is not, in the general case, an (extended) Dyck string w.r.t. the parentheses of the π_i's since the nodes in $\mathcal{N}_i$ are freely located within τ and this may lead to an unbalanced decoration string σ_τ. Therefore, it seems that the method used for TAGs cannot be generalized to MC-TAGs. The purpose of this section is to study another way to decrease the parse time complexity of MC-TAGs.

As in Section 9, for each elementary tree set $\tau = \{\tau_1, \ldots, \tau_j, \ldots, \tau_k\}$ (we assume that τ is ordered), its set of nodes is denoted by $\mathcal{N}_\tau$ and its decoration string σ_τ is the concatenation of σ_{τ_1}, $\ldots$, σ_{τ_j}, $\ldots$, σ_{τ_k} where σ_{τ_j} is the decoration string of τ_j. Each σ_{τ_j} has the form $L_{r_{\tau_j}} \ldots R_{r_{\tau_j}}$ if r_{τ_j} is the root node of τ_j. Moreover, if τ is an auxiliary tree set each σ_{τ_j} has the form $\sigma^l_{\tau_j}\sigma^r_{\tau_j}$, where $\sigma^l_{\tau_j}$ (resp. $\sigma^r_{\tau_j}$) is the prefix (resp. suffix) of the decoration string gathered before (resp. after) f_{τ_j}, the foot node of τ_j. Of course, we have $\sigma^l_{\tau_j} = L_{r_{\tau_j}} \ldots L_{f_{\tau_j}}$ and $\sigma^l_{\tau_j} = R_{f_{\tau_j}} \ldots R_{r_{\tau_j}}$.

For some $\mathcal{N} \subset \mathcal{N}_\tau$, a tuple $\vec{\sigma} = [\sigma_1, \ldots, \sigma_i, \ldots, \sigma_p]^{\mathcal{N}}$ is called a *decoration vector* of the tree set τ iff the σ_i's are non-overlapping substrings of σ_τ, occurring in that order in σ_τ and the string $\sigma_1 \ldots \sigma_i \ldots \sigma_p$ contains all the LR-variables of $\mathcal{N}$, and only those variables. More formally, we have

- $\sigma_\tau = \sigma'_0\sigma_1\sigma'_1 \ldots \sigma_i\sigma'_i \ldots \sigma_p\sigma'_p$ and
- $\eta \in \mathcal{N} \iff L_\eta, R_\eta \in \mathcal{V}_{\sigma_1 \ldots \sigma_p}$ and $\eta \in \mathcal{N}_\tau - \mathcal{N} \iff L_\eta, R_\eta \in \mathcal{V}_{\sigma'_0\sigma'_1 \ldots \sigma'_i \ldots \sigma'_p}$, if $\mathcal{V}_\sigma$ denotes the set of variables occurring in σ.

The number p of components of a decoration vector is its *size*.

As an example, if β_i is an auxiliary set of trees of cardinality k, if $\sigma_{\beta_i} = \sigma^l_{\beta_{i1}}\sigma^r_{\beta_{i1}} \ldots \sigma^l_{\beta_{ik}}\sigma^r_{\beta_{ik}}$ is its decoration string and if $\mathcal{N}_{\beta_i}$ is its set of nodes, the tuple $\vec{\sigma_{\beta_i}} = [\sigma^l_1, \sigma^r_1, \ldots, \sigma^l_k, \sigma^r_k]^{\mathcal{N}_{\beta_i}}$ is a decoration vector of β_i, for $\mathcal{N}_{\beta_i}$. Its size is $2k$. Such a decoration vector is called *initial*.

We know that any individual MCA is entirely defined by a function, its local adjunction cover ξ_j. In order to apply ξ_j, the set of nodes of its definition set $dom(\xi_j)$ must be identified. More precisely, since we deal with RCGs, the $2|dom(\xi_j)|$ LR-variables decorating the nodes of $dom(\xi_j)$ must occurred altogether within a single clause. Of course, the same condition holds for all the local adjunction covers defining a given adjunction cover. Thus we are faced with the following problem. Let β_i be an auxiliary tree set and let $\sigma_{\beta_i} = \sigma^l_{\beta_{i1}}\sigma^r_{\beta_{i1}} \ldots \sigma^l_{\beta_{ik}}\sigma^r_{\beta_{ik}}$ be its decoration string. Let $\xi_1, \ldots, \xi_j, \ldots, \xi_m$ be the local adjunction covers of ξ, an adjunction cover of β_i, and let $\Pi^\xi_\tau = \{\mathcal{N}_1, \ldots, \mathcal{N}_j, \ldots, \mathcal{N}_m\}$, $\mathcal{N}_j = dom(\xi_j)$ be the corresponding partition of $\mathcal{N}_{\beta_i}$. Starting from the initial decoration vector $\vec{\sigma_{\beta_i}} = [\sigma^l_{\beta_{i1}}, \sigma^r_{\beta_{i1}}, \ldots, \sigma^l_{\beta_{ik}}, \sigma^r_{\beta_{ik}}]^{\mathcal{N}_{\beta_i}}$, we want to build a sequence of decoration vectors from which, at some steps, the LR-variables associated with each $\mathcal{N}_j$ can be simultaneously extracted. However, we must be aware that each component of a decoration vector is also a complete argument of some predicate, and as such denotes a (contiguous) range of source symbols. Thus, if a variable is isolated in the middle of some component, its extraction will obligatory result in the creation of two arguments (one for the prefix part and the other for the suffix part). If we remember that the degree of the polynomial parse time is proportional to the predicate arities, we can easily figure out that the way and the order in which this sequence of extractions is performed is not neutral.

If we define an m-partition $\{\mathcal{N}_1, \ldots, \mathcal{N}_j, \ldots, \mathcal{N}_m\}$ of some set of nodes $\mathcal{N}$, the idea is, starting from a decoration vector $\vec{\sigma}$ for $\mathcal{N}$, to define m decoration vectors $\vec{\sigma_1}$, ..., $\vec{\sigma_j}$, ..., $\vec{\sigma_m}$. Each such decoration vector will, at turn, produce other decoration vectors until we get a decoration vector which contains all and only those variables implied in a given MCA. In order to do that, each component string σ_i of $\vec{\sigma}$ is cut into substrings in such a way that each substrings only contains LR-variables associated with the $\mathcal{N}_j$'s. Each of these substrings, becomes in turn a component of a new decoration vector $\vec{\sigma_j}$. Of course the substrings which contain the variables associated with $\mathcal{N}_j$ are components of $\vec{\sigma_j}$. More formally, let $\vec{\sigma} = [\sigma_1, \ldots, \sigma_i, \ldots, \sigma_p]^{\mathcal{N}}$ be a decoration vector for $\mathcal{N}$. If $\{\mathcal{N}_1, \ldots, \mathcal{N}_j, \ldots, \mathcal{N}_m\}$ is a partition of $\mathcal{N}$ into m non empty subsets; we call *splitting* of $\vec{\sigma}$ by $\{\mathcal{N}_1, \ldots, \mathcal{N}_j, \ldots, \mathcal{N}_m\}$ the m decoration vectors $\vec{\sigma_1} = [\sigma^1_1, \ldots, \sigma^1_{p_1}]^{\mathcal{N}_1}$,..., $\vec{\sigma_j} = [\sigma^j_1, \ldots, \sigma^j_{p_j}]^{\mathcal{N}_j}$, ..., $\vec{\sigma_m} = [\sigma^m_1, \ldots, \sigma^m_{p_m}]^{\mathcal{N}_m}$. The h^{th} component of $\vec{\sigma_j}$ is denoted σ^j_h (i.e. $\vec{\sigma_j}[h] = \sigma^j_h$). This component, which only contains LR-variables of $\mathcal{N}_j$, is a substring of some component σ_i of $\vec{\sigma}$. Let $l^j_1, \ldots, l^j_i, \ldots, l^j_p$ be respectively the numbers of substrings in $\sigma_1, \ldots, \sigma_i, \ldots, \sigma_p$ which only contains LR-variables of $\mathcal{N}_j$. We define the numbers $k^j_1, \ldots, k^j_i, \ldots, k^j_p$ by $k^j_1 = 0, k^j_2 = k^j_1 + l^j_1, \ldots, k^j_i = k^j_{i-1} + l^j_{i-1}, \ldots, k^j_p = k^j_{p-1} + l^j_{p-1}$. The components of $\vec{\sigma_j}$, which originate in σ_i, are located in $\vec{\sigma_j}$ at indexes ranging from $k^j_i + 1$ to $k^j_i + l^j_i$. Thus, the components of $\vec{\sigma_j}$, $1 \leq j \leq m$, are extracted from each σ_i, $1 \leq i \leq p$, as follows

- $\sigma_i = \nu^1_0\sigma^1_{k^1_i+1}\nu^1_1 \ldots \nu^1_{l^1_i-1}\sigma^1_{k^1_i+l^1_i}\nu^1_{l^1_i}$
- $\nu^1_0\nu^1_1 \ldots \nu^1_{l^1_i-1}\nu^1_{l^1_i} = \nu^2_0\sigma^2_{k^2_i+1}\nu^2_1 \ldots \nu^2_{l^2_i-1}\sigma^2_{k^2_i+l^2_i}\nu^2_{l^2_i}$
- ...

- $\nu_0^{j-1}\nu_1^{j-1}\ldots\nu_{l_i^{j-1}-1}^{j-1}\nu_{l_i^j}^{j-1} = \nu_0^j\sigma_{k_i^j+1}^j\nu_1^j\ldots\nu_{l_i^j-1}^j\sigma_{k_i^j+l_i^j}^j\nu_{l_i^j}^j$
- $\ldots$
- $\nu_0^{m-1}\nu_1^{m-1}\ldots\nu_{l_i^{m-1}-1}^{m-1}\nu_{l_i^m}^{m-1} = \sigma_{k_i^m+1}^m\ldots\sigma_{k_i^m+l_i^m}^m$

If $\sigma_i \in T^*$, we can note that σ_i does not contribute to any $\vec{\sigma_j}$'s. We also see that, in the general case, a splitting by some partition is not unique since a component, say $\vec{\sigma_j}[h]$, which contains several variables can always be cut in two parts.

Let ξ be a covering function for some τ, $\xi_1,\ldots,\xi_m$ its associated local covering functions and $\Pi_\tau^\xi = \{\mathcal{N}_1,\ldots,\mathcal{N}_i,\ldots,\mathcal{N}_m\}$, $\mathcal{N}_i = dom(\xi_i)$ the corresponding partition of $\mathcal{N}_\tau$. Starting, from the initial decoration vector $[\sigma_\alpha]^{\mathcal{N}_\alpha}$ if τ is an initial tree α or from $[\sigma_1^l,\sigma_1^r,\ldots,\sigma_k^l,\sigma_k^r]^{\mathcal{N}_\beta}$ if τ is an auxiliary tree set β of cardinality k, we want, by successive splittings, build m decoration vectors $[\ldots]^{\mathcal{N}_1},\ldots,[\ldots]^{\mathcal{N}_i},\ldots,[\ldots]^{\mathcal{N}_m}$ which are qualified of *final*. Of course, at each step the partition which defines a splitting must be *compatible* with the partition Π_τ^ξ. This means that the set of nodes $\mathcal{N}$ of each decoration vector $[\ldots]^{\mathcal{N}}$ must be the union of sets in Π_τ^ξ (i.e. $\exists R \subset \{i \mid 1 \leq i \leq m\}, \mathcal{N} = \cup_{i\in R}\mathcal{N}_i$). Note that, usually, a sequence of successive splittings is not unique.

Now, we are ready to state how to generate a simple $2k$-PRCG from a k-MC-TAG.

For each initial tree set α_i and for each adjunction cover ξ of α_i, we generate the clause

$$S(W) \quad \rightarrow \quad [\gamma_{\alpha_i}]_\xi^{\mathcal{N}}(W)$$

where $\mathcal{N} = \{\eta \mid \eta \in \mathcal{N}_{\alpha_i} \text{ and } \xi(\eta) \neq nil\}$ and γ_{α_i} is the decoration string σ_{α_i} of α_i in which all the LR-variables L_η and R_η associated with the nil nodes η of $\mathcal{N}_{\alpha_i} - \mathcal{N}$ have been erased. The unary predicate name $[\gamma_{\alpha_i}]_\xi^{\mathcal{N}}$ is associated with the (one component) initial decoration vector $[\gamma_{\alpha_i}]^{\mathcal{N}}$, and W is some RCG variable.

For each auxiliary tree set β_i of cardinality k, and for each adjunction cover ξ of β_i, we generate the clause

$$\beta_i(L_1,R_1,\ldots,L_k,R_k) \quad \rightarrow \quad [\gamma_{\beta_{i1}}^l,\gamma_{\beta_{i1}}^r,\ldots,\gamma_{\beta_{ij}}^l,\gamma_{\beta_{ij}}^r,\ldots,\gamma_{\beta_{ik}}^l,\gamma_{\beta_{ik}}^r]_\xi^{\mathcal{N}}(L_1,R_1,\ldots,L_k,R_k)$$

where $\mathcal{N} = \{\eta \mid \eta \in \mathcal{N}_{\beta_i} \text{ and } \xi(\eta) \neq nil\}$ and each $\gamma_{\beta_{ij}}^l$ or $\gamma_{\beta_{ij}}^r$ is the decoration string $\sigma_{\beta_{ij}}^l$ or $\sigma_{\beta_{ij}}^r$ in which the LR-variables associated with the nil nodes of $\mathcal{N}_{\beta_i} - \mathcal{N}$ have been erased. The predicate name $[\gamma_{\beta_{i1}}^l,\gamma_{\beta_{i1}}^r,\ldots,\gamma_{\beta_{ij}}^l,\gamma_{\beta_{ij}}^r,\ldots,\gamma_{\beta_{ik}}^l,\gamma_{\beta_{ik}}^r]_\xi^{\mathcal{N}}$ of arity $2k$ is associated with the ($2k$ component) initial decoration vector $[\gamma_{\beta_{i1}}^l,\gamma_{\beta_{i1}}^r,\ldots,\gamma_{\beta_{ij}}^l,\gamma_{\beta_{ij}}^r,\ldots,\gamma_{\beta_{ik}}^l,\gamma_{\beta_{ik}}^r]^{\mathcal{N}}$, and the L_i's and R_i's are RCG variables.

For each not yet defined predicate name $[\sigma_1,\ldots,\sigma_i,\ldots,\sigma_p]_\xi^{\mathcal{N}}$, associated with the decoration vector $[\sigma_1,\ldots,\sigma_i,\ldots,\sigma_p]^{\mathcal{N}}$, if $\{\mathcal{N}_1,\ldots,\mathcal{N}_j,\ldots,\mathcal{N}_m\}$, $m \geq 2$ is a partition of $\mathcal{N}$, which is compatible with ξ, we generate the clause

$$[\sigma_1,\ldots,\sigma_i,\ldots,\sigma_p]_\xi^{\mathcal{N}}(\gamma_1,\ldots,\gamma_i,\ldots,\gamma_p) \quad \rightarrow \quad \psi_1\ldots\psi_j\ldots\psi_m$$

if the splitting of $[\sigma_1,\ldots,\sigma_i,\ldots,\sigma_p]^{\mathcal{N}}$ by $\{\mathcal{N}_1,\ldots,\mathcal{N}_j,\ldots,\mathcal{N}_m\}$ gives the m decoration vectors $[\sigma_1^1,\ldots,\sigma_{p_1}^1]^{\mathcal{N}_1}$, ...,$[\sigma_1^j,\ldots,\sigma_{p_j}^j]^{\mathcal{N}_j}$, ..., $[\sigma_1^m,\ldots,\sigma_{p_m}^m]^{\mathcal{N}_m}$. Each predicate call ψ_j has the form

$[\gamma_1^j, \ldots, \gamma_{p_j}^j]_\xi^{\mathcal{N}_j}(\underline{\gamma_1^j}, \ldots, \underline{\gamma_{p_j}^j})$, where γ_h^j, $1 \leq h \leq p_j$ is the string σ_h^j in which the leading and trailing elements in T^+ have been erased and the underlined terms $\underline{\gamma_h^j}$, $1 \leq h \leq p_j$ denote RCG variables. In the LHS, each argument γ_i is the string σ_i in which each substring such as γ_h^j as been replaced by the corresponding RCG variable $\underline{\gamma_h^j}$.

For each not yet defined predicate name $[\sigma_1, \ldots, \sigma_p]_\xi^{\mathcal{N}}$, if $\mathcal{N}$ are nodes at which a single MCA of β_i defined by ξ takes place, we generate

$$[\sigma_1, \ldots, \sigma_p]_\xi^{\mathcal{N}}(\sigma_1, \ldots, \sigma_p) \quad \rightarrow \quad \beta_i(L_1, R_1, \ldots, L_j, R_j, \ldots, L_k, R_k)$$

if the cardinality of the auxiliary tree set β_i is k, and if $\forall \eta \in \mathcal{N}$ we have $\xi(\eta) = \beta_{ij} \Longrightarrow L_j = L_\eta, R_j = R_\eta$.

In this paper, we assume that the previous transformation gives a simple PRCG which is equivalent to the original MC-TAG. We will now address the parse time complexity of the resulting RCG.

When a splitting of $[\sigma_1, \ldots, \sigma_p]^{\mathcal{N}}$ by $\Pi = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_q\}$ gives the q decoration vectors $[\sigma_1^1, \ldots, \sigma_{p_1}^1]^{\mathcal{N}_1}$, $[\sigma_1^2, \ldots, \sigma_{p_2}^2]^{\mathcal{N}_2}$, ..., $[\sigma_1^q, \ldots, \sigma_{p_q}^q]^{\mathcal{N}_q}$ the number of variables in the associated clause is $p_1 + p_2 + \ldots + p_q$ and the corresponding number of free bounds is $p + p_1 + p_2 + \ldots + p_q$. Of course, if $[\sigma_1, \ldots, \sigma_p]^{\mathcal{N}}$ is an initial decoration vector and if Π is the partition associated with the covering function ξ, we are in the unoptimized generation case and $p + p_1 + p_2 + \ldots + p_q = 2(k+v)$, where v is the maximum number of nonterminal node hosting a non nil adjunction constraint. The purpose of the previous generation strategy is to find a splitting sequence, which starts from some initial decoration vector and leads to final decoration vectors, while minimizing (at each step) the number of free bounds. However, it must be clear that, in some cases, this minimum number of free bounds can be the maximum value $2(k+v)$.[23] However, fairly often, several variables can be grouped together within the same decoration vector component (from which, altogether, they will be extracted later on during another splitting), leading to a decreased parse time.

If, as in the TAG case, we want to get a worst case parse time which does not depend upon the number of adjunction nodes within an elementary tree set, we must impose some restrictions on the initial k-MC-TAG. First we will assume that the associated simple PRCG is of arity $2k$. As already noticed, this is not a restriction on the input MC-TAG (in each splitting, the sizes of the resulting decoration vectors are bounded by $2k$), but this restricts the number of splitting sequences and may thus not lead to an optimal solution. Second, we limit the total number of components used at each splitting step. Since for a k-MC-TAG we have to isolate at least $2k$ LR-variables, this total number of components must be at least $2k$ (if we assume that each tree of an auxiliary tree set of cardinality k adjoined at spinal nodes of k different trees), and thus, is of the form $2k + c$, $c \geq 0$. In this case the parse time complexity has the form $\mathcal{O}(n^{4k+c})$ and we say that we have an MC-TAG in c-split form. It seems that the first interesting class within this c-split form hierarchy is for $c = 2$. In this case, all the arguments of a predicate definition of arity $2k$ contain a single variable, except that there is

[23] Imagine a TAG ($k = 1$) in which all elementary trees contain a single nonterminal node which can host a non nil adjunction ($v = 1$), if moreover, in some auxiliary tree β, this unique adjunction node η is a spinal node, the LHS of the associated clause has the form $\beta(uL_\eta u', vR_\eta v')$, $u, u', v, v' \in T^*$ whose number of free bounds is 4, equal to the value of $2(k+v)$.

either one argument with three variables or there are two arguments with two variables each. Note that the parse time complexity of a k-MC-TAG in 2-split form is $\mathcal{O}(n^{4k+2})$.[24]

Example 3 (concluded) *It is not difficult to see that our optimized generation strategy, when applied to our example, exactly gives the same simple PRCG. In fact, the original grammar is a 3-MC-TAG in 0-split form, and its worst case parse time is thus $\mathcal{O}(n^{12})$. However, we can remark that on the one hand, all the R-variables are bounded to an empty range, and, on the other hand, that any instantiation of a β_i-clause succeeds (or fails) in a time linear with the sum of the sizes of its arguments, thus, we can easily figure out that the 3-copy language can be parsed in cubic time. This parse time even decreases to linear if we remark that the three variables $L_{\alpha.1}$, $L_{\alpha.2}$ and $L_{\alpha.3}$ must be bound to ranges of equal sizes.*

11 Conclusion

In this paper, we first proposed two methods to implement a TAG parser in $\mathcal{O}(n^6)$ time. These algorithms use RCGs, a powerful high level syntactic description formalism, as an intermediate object language. Since [Boullier, 1998b], we know that RCGs can be used to implement parsers for TAGs. However, the $\mathcal{O}(n^6)$ bound was only reached with a restricted form of adjunction constraints and when the initial TAG was in some normal form. In particular, this normal form assumes that elementary trees are in a binary branching form, form in which the original structure has disappeared. At the contrary, the algorithms presented here, work for completely unrestricted TAGs though their corresponding parsers still work in $\mathcal{O}(n^6)$ time at worst.

For a linguistically significant restricted form of TAGs that can be parsed in $\mathcal{O}(n^5)$, we have shown that its RCG version can also be parsed in $\mathcal{O}(n^5)$ time, whether we start from the intermediate representation introduced in [Satta and Schuler, 1998], or directly from the elementary trees representation. We have even noticed that our algorithm for unrestricted TAGs can be modified to take into account the restricted form, and results in a single translation algorithm which produces RCGs which can be parsed in $\mathcal{O}(n^5)$ at worst when the original TAG is in restricted form. The RCG formalism even guarantees that restricted parts of an unrestricted TAG are processed in $\mathcal{O}(n^5)$ at worst.

Afterwards, we have shown that any set-local MC-TAG with unrestricted multicomponent adjunction constraints can be translated into an equivalent simple PRCG. This PRCG, in turn, as any other RCG, can be parsed in polynomial time. However, in the general case, the degree of this polynomial depends both on the maximum number k of elementary trees in a tree set and on the maximum number v of adjunction nodes in a tree set: we showed that an MC-TAG can be parsed at worst in $\mathcal{O}(n^{2(k+v)})$ time. In order to release from the v parameter, we define an optimized generation algorithm which always succeeds and tries to define some value c leading to a grammar in the so-called c-split form. For k-MC-TAGs in c-split form, the parse time of its equivalent simple PRCG is $\mathcal{O}(n^{4k+c})$. This c parameter depends on the relative places where multicomponent adjunctions can take place.

[24] The linguistic relevance of such a restriction will not be examined. However, we can remark that any unrestricted TAG is a 1-MC-TAG in 2-split form.

If we assume that, in order to be (linguistically) interesting, an MC-TAG must be at least as powerful as a TAG, the first subclass, in the split form hierarchy, corresponds to $c = 2$, for we have noticed that TAGs and 1-MC-TAGs in 2-split form are equivalent. Of course, in this case, both parsers work in $\mathcal{O}(n^6)$ time at worst. However, the linguistic relevance of the split form hierarchy still has to be demonstrated though it can easily be shown that multiple agreements of degree k (i.e. $\{a_1^n b_1^n \dots a_k^n b_k^n \mid n \geq 1, k \geq 2\}$) can be defined by a $(k-1)$-MC-TAG in 0-split form and that duplication of degree k (i.e. $\{w^k \mid w \in \{a,b\}^*, k \geq 2\}$) can also be defined by a k-MC-TAG in 0-split form.[25]

The usage of RCGs as an intermediate structure may result in several advantages. First, since the RCG formalism is simple, the transformation proposed in this paper gives another view of the (rather complicated) adjunction or multicomponent adjunction mechanism and may help to understand when and why they are costly to implement. Second, since RCGs can be efficiently implemented, we are convinced that these methods are good candidates for practical TAG or MC-TAGs implementations.

References

[Abeillé, 1994] Abeillé A. (1994). Syntax or Semantics? handling Nonlocal Dependencies with MCTAGs or Synchronous TAGs In *Computational Intelligence*, Vol. 10, No. 4, pages 471–485.

[Becker, Joshi and Rambow, 1991] Becker T., Joshi A. and Rambow O. (1991). Long distance scrambling and tree adjoining grammars. In *Proceedings of the fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL'91)*, pages 21–26.

[Boullier, 1996] Boullier P. (1996). Another Facet of LIG Parsing In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, University of California at Santa Cruz, California, 24–27 June, pages 87–94. See also *Research Report No 2858* at `http://www.inria.fr/RRRT/RR-2858.html`, INRIA-Rocquencourt, France, Apr. 1996, 22 pages.

[Boullier, 1998a] Boullier P. (1998). Proposal for a Natural Language Processing Syntactic Backbone. In *Research Report No 3342* at `http://www.inria.fr/RRRT/RR-3342.html`, INRIA-Rocquencourt, France, Jan. 1998, 41 pages.

[Boullier, 1998b] Boullier P. (1998). A Generalization of Mildly Context-Sensitive Formalisms. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+4)*, University of Pennsylvania, Philadelphia, PA, 1–3 August, pages 17–20.

[Boullier, 1999a] Boullier P. (1999). Chinese Numbers, MIX, Scrambling, and Range Concatenation Grammars In *Proceedings of the 9th Conference of the Europwean Chapter of the Association for Computational Linguistics (EACL'99)*, Bergen, Norway, June 8–12.

[25] Recall that cross agreements of degree k (i.e. $\{a_1^{n_1} \dots a_k^{n_k} b_1^{n_1} \dots b_k^{n_k} \mid n_i \geq 1\}$) is a TAL.

[Boullier, 1999b] Boullier P. (1999). On TAG Parsing In 6*ème conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'99)*, Cargèse, Corsica, France, July 12-17.

[Boullier, 1999c] Boullier P. (1999). On Multicomponent TAG Parsing In 6*ème conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'99)*, Cargèse, Corsica, France, July 12-17.

[Gazdar, 1985] Gazdar G. (1985). Applicability of Indexed Grammars to Natural Languages. In *Technical Report CSLI-85-34*, Center for Study of Language and Information, 1985.

[Groenink, 1997] Groenink A. (1997). SURFACE WITHOUT STRUCTURE Word order and tractability issues in natural language analysis. PhD thesis, Utrecht University, The Netherlands, Nov. 1977, 250 pages.

[Joshi, Levy, and Takahashi, 1975] Joshi A., Levy L. and Takahashi M. (1975). Tree adjunct grammars. In *Journal of Computer ans System Sciences*, 10, pages 136–163.

[Joshi, 1985] Joshi A. (1985). How much context-sensitivity is necessary for characterizing structural descriptions — Tree Adjoining Grammars. In *Natural Language Processing — Theoritical, Computational and Psychological Perspective*, D. Dowty, L. Karttunen, and A. Zwicky, editors, Cambridge University Press, New-York, NY.

[Joshi, 1987] Joshi A. (1987). An Introduction to Tree Adjoining Grammars. In *Mathematics of Language*, Manaster-Ramer, A., editors, John Benjamins, Amsterdam, pages 87–114.

[Kroch and Joshi, 1986] Kroch A. and Joshi A. (1986). Analyzing Extraposition in a Tree Adjoining Grammar. In *Discontinuous constituents, syntax and semantics*, Edited by Huck and Ojeda. Academic Press, New York, vol. 20, pages 107–149.

[Pollard, 1984] Pollard C. (1984). Generalized Phrase Structure Grammars, Head Grammars and Natural Language. PhD thesis, Stanford University.

[Rambow and Lee, 1994] Rambow O. and Lee Y.-S.. (1994). Word Order Variation and Tree Adjoining Grammar. In *Computational Intelligence*, Vol. 10, No. 4, pages 386–400.

[Rounds, 1988] Rounds W. (1988). LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. In *ACL Computational Linguistics*, Vol. 14, No. 4, pages 1–9.

[Satta, 1994] Satta G. (1994). Tree adjoining grammars parsing and boolean matrix multiplication. In *Computational Linguistics*, 20(2), pages 173–192.

[Satta and Schuler, 1998] Satta G. and Schuler W. (1998). Restrictions on Tree Adjoining Languages. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL'98)*, Université de Montréal, Montréal, Québec, Canada, 10–14 August, vol. II, pages 1176–1182.

[Schabes and Shieber, 1994] Schabes Y. and Shieber S. (1994). An Alternative Conception of Tree-Adjoining Derivation. In *ACL Computational Linguistics*, Vol. 20, No. 1, pages 91–124.

[Schabes and Waters, 1994] Schabes Y. and Waters R. (1994). Tree Insertion Grammar: A Cubic-Time Parsable Formalism That Lexicalizes Context-Free Grammar Without Changing the Trees Produced In *Technical Report*, TR-94-13, Mitsubishi Electric Research Laboratories, June, 35 pages.

[Seki, Matsumura, Fujii, and Kasami, 1991] Seki H., Matsumura T., Fujii M. and Kasami T. (1991). On multiple context-free grammars. In *Theoretical Computer Science*, Elsevier Science, 88, pages 191–229.

[Steedman, 1987] Steedman M. (1987). Combinatory grammars and parasitic gaps. In *Natural language and Linguistic Theory*, 1987.

[Vijay-Shanker, 1987] Vijay-Shanker K. (1987). A study of tree adjoining grammars. *PhD thesis*, University of Pennsylvania, Philadelphia, PA.

[Vijay-Shanker, Weir, and Joshi, 1987] Vijay-Shanker K., Weir D. and Joshi A. (1987). 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, pages 104–111.

[Vijay-Shanker and Weir, 1993] Vijay-Shanker K. and Weir D. (1993). The Used of Shared Forests in Tree Adjoining Grammar Parsing. In *Proceedings of the 6th Conference of the European Chapter of the Association for Computational Linguistics (EACL'93)*, Utrecht, The Netherlands, pages 384–393.

[Vijay-Shanker and Weir, 1994a] Vijay-Shanker K. and Weir D. (1994). The equivalence of four extensions of context-free grammars. In *Math. Systems Theory*, Vol. 27. pages 511–546

[Vijay-Shanker and Weir, 1994b] Vijay-Shanker K. and Weir D. (1994). Parsing some constrained grammar formalisms. In *ACL Computational Linguistics*, Vol. 19, No. 4, pages 591–636.

[Vijay-Shanker, Weir and Rambow, 1995] Vijay-Shanker K., Weir D. and Rambow O. (1995). Parsing D-Tree Grammars. In *Proceedings of the fourth international workshop on parsing technologies (IWPT'95)*, Prague and Karlovy Vary, Czech Republic, pages 252–259.

[Weir, 1988] Weir D. (1988). Characterizing Mildly Context-Sensitive Grammar Formalisms. In *PhD thesis*, University of Pennsylvania, Philadelphia, PA.

Unit´e de recherche INRIA Lorraine, Technop^ole de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rh^one-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399