TuLiPA: A Syntax-Semantics Parsing Environment for Mildly Context-Sensitive Formalisms

Yannick Parmentier
CNRS - LORIA
Nancy Université
F-54506, Vandœuvre, France
parmenti@loria.fr

Laura Kallmeyer
SFB 441
Universität Tübingen
D-72074, Tübingen, Germany
lk@sfs.uni-tuebingen.de

Wolfgang Maier
SFB 441
Universität Tübingen
D-72074, Tübingen, Germany
wo.maier@uni-tuebingen.de

Timm Lichte
SFB 441
Universität Tübingen
D-72074, Tübingen, Germany
timm.lichte@uni-tuebingen.de

Johannes Dellert
SFB 441 - SfS
Universität Tübingen
D-72074, Tübingen, Germany
jdellert@sfs.uni-tuebingen.de

Abstract

In this paper we present a parsing architecture that allows processing of different mildly context-sensitive formalisms, in particular Tree-Adjoining Grammar (TAG), Multi-Component Tree-Adjoining Grammar with Tree Tuples (TT-MCTAG) and *simple* Range Concatenation Grammar (RCG). Furthermore, for tree-based grammars, the parser computes not only syntactic analyses but also the corresponding semantic representations.

1 Introduction

The starting point of the work presented here is the aim to implement a parser for a German TAG-based grammar that computes syntax and semantics. As a grammar formalism for German we chose a multicomponent extension of TAG called TT-MCTAG (Multicomponent TAG with Tree Tuples) which has been first introduced by Lichte (2007). With some additional constraints, TT-MCTAG is mildly context-sensitive (MCS) as shown by Kallmeyer and Parmentier (2008).

Instead of implementing a specific TT-MCTAG parser we follow a more general approach by using Range Concatenation Grammars (RCG) as a pivot formalism for parsing MCS languages. Indeed the generative capacity of RCGs lies beyond MCS, while they stay parsable in polynomial time (Boullier, 1999). In this context, the TT-MCTAG (or TAG) is transformed into a strongly equivalent RCG that is then used for parsing. We have implemented the conversion into RCG, the RCG parser and the retrieval of the corresponding TT-MCTAG analyses. The parsing architecture comes with graphical input and output interfaces, and an XML export of the result of parsing. It is called TuLiPA (for "Tübingen Linguistic Parsing Architecture") and is freely available under the GPL.[1] Concretely, TuLiPA processes TT-MCTAGs and TAGs encoded in the XML format of the XMG (eXtensible MetaGrammar) system of Duchier et al. (2004).

In this paper, we present this parsing architecture focusing on the following aspects: first, we introduce the TT-MCTAG formalism (section 2). Then, we present successively the RCG formalism (section 3) and the conversion of TT-MCTAG into RCG (section 4). Section 5 shows how RCG is parsed in practice. Eventually, we present the retrieval of TT-MCTAG derivation structures (section 6), the computation of semantic representations (section 7) and optimizations that have been added to speed up parsing (section 8).

2 TT-MCTAG

TT-MCTAGs (Lichte, 2007) are multicomponent TAGs (MCTAG) where the elementary tree sets consist of one lexicalized tree γ, the *head tree* and a set of auxiliary trees $\beta_1, ..., \beta_n$, the *argument trees*. We write these sets as tuples $\langle \gamma, \{\beta_1, ..., \beta_n\} \rangle$. During derivation, the argument trees have to attach to their head, either directly or indirectly via *node sharing*. The latter means that they are linked by a chain of root-adjunctions to a tree adjoining to their head.

[1] http://sourcesup.cru.fr/tulipa/

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

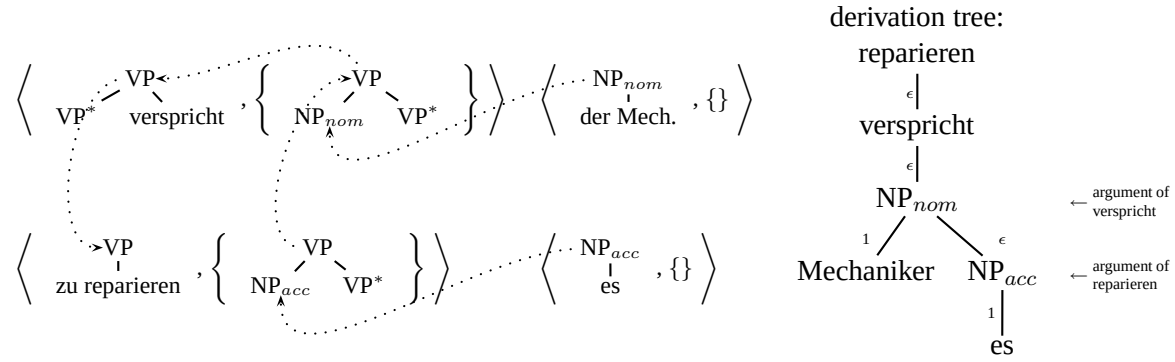


Figure 1: TT-MCTAG analysis of (1)

Definition 1 (TT-MCTAG) *An MCTAG $G = \langle I, A, N, T, \mathcal{A}\rangle$ is a TT-MCTAG iff*

1. *every $\Gamma \in \mathcal{A}$ has the form $\{\gamma, \beta_1, \ldots, \beta_n\}$ where γ contains at least one leaf with a terminal label, the* head tree*, and $\beta_1, \ldots, \beta_n$ are auxiliary trees, the* argument trees*. We write such a set as a tuple $\langle \gamma, \{\beta_1, \ldots, \beta_n\}\rangle$.*

2. *A derivation tree D for some $t \in L(\langle I, A, N, T\rangle)$ is licensed as a TAG derivation tree in G iff D satisfies the following conditions (MC) ("multicomponent condition") and (SN-TTL) ("tree-tuple locality with shared nodes"):*

   (a) **(MC)** *There are k pairwise disjoint instances $\Gamma_1, \ldots, \Gamma_k$ of elementary tree sets from $\mathcal{A}$ for some $k \geq 1$ such that $\bigcup_{i=1}^{k} \Gamma_i$ is the set of node labels in D.*

   (b) **(SN-TTL)** *for all nodes $n_0, n_1, \ldots, n_m$, $m > 1$, in D with labels from the same elementary tree tuple such that n_0 is labelled by the head tree: for all $1 \leq i \leq m$: either $\langle n_0, n_i\rangle \in \mathcal{P}_D$[2] or there are $n_{i,1}, \ldots, n_{i,k}$ with auxiliary tree labels such that $n_i = n_{i,k}$, $\langle n_0, n_{i,1}\rangle \in \mathcal{P}_D$ and for $1 \leq j \leq k-1$: $\langle n_{i,j}, n_{i,j+1}\rangle \in \mathcal{P}_D$ where this edge is labelled with ϵ.*

TT-MCTAG has been proposed to deal with free word order languages. An example from German is shown in Fig. 1. Here, the NP$_{nom}$ auxiliary tree adjoins directly to *verspricht* (its head) while the NP$_{acc}$ tree adjoins to the root of a tree that adjoins to the root of a tree that adjoins to *reparieren*.

For a more extended account of German word order using TT-MCTAG see Lichte (2007) and Lichte and Kallmeyer (2008).

TT-MCTAG can be further restricted, such that at each point of the derivation the number of pending β-trees is at most k. This subclass is also called k-TT-MCTAG.

Definition 2 (k-TT-MCTAG) *A TT-MCTAG $G = \langle I, A, N, T, \mathcal{A}\rangle$ is of rank k (or a k-TT-MCTAG for short) iff for each derivation tree D licensed in G:*

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

TT-MCTAG in general are NP-complete (Søgaard et al., 2007) while k-TT-MCTAG are MCS (Kallmeyer and Parmentier, 2008).

3 RCG as a pivot formalism

The central idea of our parsing strategy is to use RCG (Boullier, 1999; Boullier, 2000) as a pivot formalism.

Definition 3 (RCG) *A RCG is a tuple $G = \langle N, T, V, S, P\rangle$ such that a) N is an alphabet of predicates of fixed arities; b) T and V are disjoint alphabets of terminals and of variables; c) $S \in N$ is the start predicate (of arity 1) and d) P is a finite set of* clauses

$A_0(x_{01}, \ldots, x_{0a_0}) \rightarrow \epsilon$,

or

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

$A_0(x_{01}, \ldots, x_{0a_0}) \rightarrow$
$\quad A_1(x_{11}, \ldots, x_{1a_1}) \ldots A_n(x_{n1}, \ldots, x_{na_n})$
with $n \geq 1$, $A_i \in N, x_{ij} \in (T \cup V)^*$ *and* a_i *being the arity of* A_i.

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

When applying a clause with respect to a string $w = t_1 \ldots t_n$, the arguments in the clause are instantiated with substrings of w, more precisely with the corresponding ranges.[4] The instantiation of a clause maps all occurrences of a $t \in T$ in the clause to an occurrence of a t in w and consecutive elements in a clause argument are mapped to consecutive ranges.

If a clause has an instantiation wrt w, then, in one derivation step, the left-hand side of this instantiation can be replaced with its right-hand side. The language of an RCG G is $L(G) = \{w \mid S(\langle 0, |w| \rangle) \stackrel{*}{\Rightarrow} \epsilon \text{ wrt } w\}$.

A sample RCG is shown in Fig. 2.

RCG: $G = \langle \{S, A, B\}, \{a, b\}, \{X, Y, Z\}, S, P \rangle$
$S(X\,Y\,Z) \rightarrow A(X, Z)\,B(Y)$,
$A(a\,X, a\,Y) \rightarrow A(X, Y)$, $A(\epsilon, \epsilon) \rightarrow \epsilon$,
$B(b\,X) \rightarrow B(X)$, $B(\epsilon) \rightarrow \epsilon$.
Input: $w = aabaa$.
Derivation:
$S(XYZ) \rightarrow A(X,Z)B(Y)$
$\langle 0,2\rangle\langle 2,3\rangle\langle 3,5\rangle \;\; \langle 0,2\rangle\langle 3,5\rangle\langle 2,3\rangle$
$\quad aa \quad b \quad aa \quad\quad aa \quad aa \quad b$
yields $S(\langle 0,5\rangle) \Rightarrow A(\langle 0,2\rangle, \langle 3,5\rangle)B(\langle 2,3\rangle)$.
$B(bX) \rightarrow B(X)$ and $B(\epsilon) \rightarrow \epsilon$
$\langle 2,3\rangle\langle 3,3\rangle\langle 3,3\rangle$
$\quad b \quad\; \epsilon \quad\; \epsilon$
yield $A(\langle 0,2\rangle, \langle 3,5\rangle)B(\langle 2,3\rangle) \Rightarrow$
$A(\langle 0,2\rangle, \langle 3,5\rangle)B(\langle 3,3\rangle) \Rightarrow A(\langle 0,2\rangle, \langle 3,5\rangle)$.
$A(aXaY) \rightarrow A(X,Y)$
$\langle 0,1\rangle\langle 1,2\rangle\langle 3,4\rangle\langle 4,5\rangle\langle 1,2\rangle\langle 4,5\rangle$
$\quad a \quad\; a \quad\; a \quad\; a \quad\; a \quad\; a$
yields $A(\langle 0,2\rangle, \langle 3,5\rangle) \Rightarrow A(\langle 1,2\rangle, \langle 4,5\rangle)$.
$A(aXaY) \rightarrow A(X,Y)$ and $A(\epsilon, \epsilon) \rightarrow \epsilon$
$\langle 1,2\rangle\langle 2,2\rangle\langle 4,5\rangle\langle 5,5\rangle\langle 2,2\rangle\langle 5,5\rangle$
$\quad a \quad\; \epsilon \quad\; a \quad\; \epsilon \quad\; \epsilon \quad\; \epsilon$
yield $A(\langle 1,2\rangle, \langle 4,5\rangle) \Rightarrow A(\langle 2,2\rangle, \langle 5,5\rangle) \Rightarrow \epsilon$

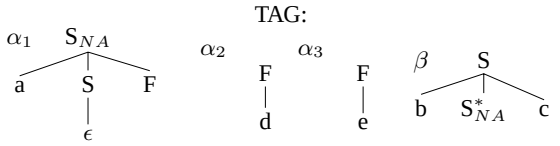
Figure 2: Sample RCG

[3]The negative variant allows for negative predicate calls of the form $\overline{A(\alpha_1, \ldots, \alpha_n)}$. Such a predicate is meant to recognize the complement language of its positive counterpart, see Boullier (2000).

[4]A range $\langle i, j \rangle$ with $0 \leq i < j \leq n$ corresponds to the substring between positions i and j, i.e., to $t_{i+1} \ldots t_j$.

4 Transforming TT-MCTAG into RCG

The transformation of a given k-TT-MCTAG into a strongly equivalent simple RCG is an extension of the TAG-to-RCG transformation proposed by Boullier (1999). The idea of the latter is the following: the RCG contains predicates $\langle\alpha\rangle(X)$ and $\langle\beta\rangle(L, R)$ for initial and auxiliary trees respectively. X covers the yield of α and all trees added to α, while L and R cover those parts of the yield of β (including all trees added to β) that are respectively to the left and the right of the foot node of β. The clauses in the RCG reduce the argument(s) of these predicates by identifying those parts that come from the elementary tree α/β itself and those parts that come from one of the elementary trees added by substitution or adjunction. An example is shown in Fig. 3.



Equivalent RCG:
$S(X) \rightarrow \langle\alpha_1\rangle(X) \mid \langle\alpha_2\rangle(X) \mid \langle\alpha_3\rangle(X)$
$\langle\alpha_1\rangle(aF) \rightarrow \langle\alpha_2\rangle(F) \mid \langle\alpha_3\rangle(F)$
$\langle\alpha_1\rangle(aB_1B_2F) \rightarrow$
$\quad \langle\beta\rangle(B_1, B_2)\langle\alpha_2\rangle(F) \mid \langle\beta\rangle(B_1, B_2)\langle\alpha_3\rangle(F)$
$\langle\beta\rangle(B_1b, cB_2) \rightarrow \langle\beta\rangle(B_1, B_2)$
$\langle\alpha_2\rangle(d) \rightarrow \epsilon \quad \langle\alpha_3\rangle(e) \rightarrow \epsilon \quad \langle\beta\rangle(b, c) \rightarrow \epsilon$

Figure 3: A TAG and an equivalent RCG

For the transformation from TT-MCTAG into RCG we use the same idea. There are predicates $\langle\gamma ...\rangle$ for the elementary trees (not the tuples) that characterize the contribution of γ. We enrich these predicates in a way that allows to keep track of the "still to adjoin" argument trees and constrain thereby further the RCG clauses. The pending arguments are encoded in a list that is part of the predicate name. The yield of a predicate corresponding to a tree γ contains not only γ and its arguments but also arguments of predicates that are higher in the derivation tree and that are adjoined below γ via node sharing. In addition, we use branching predicates adj and sub that allow computation of the possible adjunctions or substitutions at a given node in a separate clause.

As an example see Fig. 4. The first clause states that the yield of the initial α_{rep} consists of the left and right parts of the root-adjoining tree wrapped around *zu reparieren*. The adj predicate takes care

$\langle \alpha_{rep}, \emptyset \rangle (L\ zu\ reparieren\ R) \rightarrow \langle adj, \alpha_{rep}, \epsilon, \{\beta_{acc}\} \rangle (L, R)$
$\langle adj, \alpha_{rep}, \epsilon, \{\beta_{acc}\} \rangle (L, R) \rightarrow \langle \beta_{acc}, \emptyset \rangle (L, R) \mid \langle \beta_v, \{\beta_{acc}\} \rangle (L, R)$
$\langle \beta_{acc}, \emptyset \rangle (L\ X, R) \rightarrow \langle adj, \beta_{acc}, \epsilon, \emptyset \rangle (L, R) \langle sub, \beta_{acc}, 1 \rangle (X)$
$\langle sub, \beta_{acc}, 1 \rangle (X) \rightarrow \langle \alpha_{es}, \emptyset \rangle (X) \qquad \langle \alpha_{es}, \emptyset \rangle (es) \rightarrow \epsilon$
$\langle \beta_v, \{\beta_{acc}\} \rangle (L, verspricht\ R) \rightarrow \langle adj, \beta_v, \epsilon, \{\beta_{nom}, \beta_{acc}\} \rangle (L, R)$

Figure 4: Some clauses of the RCG corresponding to the TT-MCTAG in Fig. 1

of the adjunction at the root (address ϵ). It states that the list of pending arguments contains already β_{acc}, the argument of α_{rep}. According to the second clause, we can adjoin either β_{acc} (while removing it from the list of pending arguments) or some new auxiliary tree β_v.

The general construction goes as follows: We define the decoration string σ_γ of an elementary tree γ as in Boullier (1999): each internal node has two variables L and R and each substitution node has one variable X (L and R represent the left and right parts of the yield of the adjoined tree and X represents the yield of a substituted tree). In a top-down-left-to-right traversal the left variables are collected during the top-down traversal, the terminals and variables of substitution nodes are collected while visiting the leaves and the right variables are collected during bottom-up traversal. Furthermore, while visiting a foot node, a separating "," is inserted. The string obtained in this way is the decoration string.

1. We add a start predicate S and clauses $S(X) \rightarrow \langle \alpha, \emptyset \rangle (X)$ for all $\alpha \in I$.

2. For every $\gamma \in I \cup A$: Let L_p, R_p be the left and right symbols in σ_γ for the node at position p if this is not a substitution node. Let X_p be the symbol for the node at position p if this is a substitution node. We assume that $p_1, \ldots, p_k$ are the possible adjunction sites, $p_{k+1}, \ldots, p_l$ the substitution sites in γ. Then the RCG contains all clauses
$\langle \gamma, LPA \rangle (\sigma_\gamma) \rightarrow$
$\langle adj, \gamma, p_1, LPA_{p_1} \rangle (L_{p_1}, R_{p_1})$
$\ldots \langle adj, \gamma, p_k, LPA_{p_k} \rangle (L_{p_k}, R_{p_k})$
$\langle sub, \gamma, p_{k+1} \rangle (X_{p_{k+1}}) \ldots \langle sub, \gamma, p_l \rangle (X_{p_l})$
such that
   - If $LPA \neq \emptyset$, then $\epsilon \in \{p_1, \ldots, p_k\}$ and $LPA \subseteq LPA_\epsilon$, and
   - $\bigcup_{i=0}^{k} LPA_{p_i} = LPA \cup \Gamma(\gamma)$ where $\Gamma(\gamma)$ is either the set of arguments of γ (if γ is a head tree) or (if γ is an argument itself), the empty set.

3. For all predicates $\langle adj, \gamma, dot, LPA \rangle$ the RCG contains all clauses
$\langle adj, \gamma, dot, LPA \rangle (L, R) \rightarrow$
$\langle \gamma', LPA' \rangle (L, R)$
such that γ' can be adjoined at position dot in γ and
   - either $\gamma' \in LPA$ and $LPA' = LPA \setminus \{\gamma'\}$,
   - or $\gamma' \notin LPA$, γ' is a head (i.e., a head tree), and $LPA' = LPA$.

4. For all predicates $\langle adj, \gamma, dot, \emptyset \rangle$ where dot in γ is no OA-node, the RCG contains a clause $\langle adj, \gamma, dot, \emptyset \rangle (\epsilon, \epsilon) \rightarrow \epsilon$.

5. For all predicates $\langle sub, \gamma, dot \rangle$ and all γ' that can be substituted into position dot in γ the RCG contains a clause
$\langle sub, \gamma, dot \rangle (X) \rightarrow \langle \gamma', \emptyset \rangle (X)$.

5 RCG parsing

The input sentence is parsed using the RCG computed from the input TT-MCTAG via the conversion algorithm introduced in the previous section. Note that the TT-MCTAG to RCG transformation is applied to a subgrammar selected from the input sentence[5], for the cost of the conversion is proportional to the size of the grammar (all licensed adjunctions have to be computed while taking into account the state of the list of pending arguments).[6]

The RCG parsing algorithm we use is an extension of Boullier (2000). This extension concerns (i) the production of a shared forest and (ii) the use of constraint-based techniques for performing some subtask of RCG parsing.

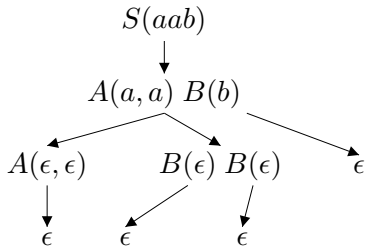
[5] In other terms, the RCG conversion is done *on-line*.

[6] We do not have a proof of complexity of the conversion algorithm yet, but we conjecture that it is exponential in the size of the grammar since the adjunctions to be predicted depend on the adjunctions predicted so far and on the auxiliary trees adjoinable at a given node.

RCG:

$$
\begin{array}{lllll}
C_0 & S(XYZ) & \rightarrow & A(X,Y)B(Z) \\
C_1 & A(aX,aY) & \rightarrow & A(X,Y) \\
C_2 & A(aX,aY) & \rightarrow & B(X)B(Y) \\
C_3 & B(\epsilon) & \rightarrow & \epsilon \\
C_4 & B(b) & \rightarrow & \epsilon \\
C_5 & A(\epsilon,\epsilon) & \rightarrow & \epsilon
\end{array}
$$

RCG Derivation wrt aab:



RCG shared forest:

$$
\begin{array}{lll}
C_0(X := a, Y := a, Z := b) & \rightarrow & (\, C_1(X := \epsilon, Y := \epsilon) \vee C_2(X := \epsilon, Y := \epsilon)\,) \quad \wedge \quad C_4 \\
C_1(X := \epsilon, Y := \epsilon) & \rightarrow & C_5 \\
C_2(X := \epsilon, Y := \epsilon) & \rightarrow & C_3 \wedge C_3
\end{array}
$$

Figure 5: RCG derivation and corresponding shared forest.

5.1 Extracting an RCG shared forest

Boullier (2000) proposes a recognition algorithm relying on two interdependent functions: one for instantiating predicates, and one for instantiating clauses. Recognition is then triggered by asking for the instantiation of the start predicate with respect to the input string. An interesting feature of Boullier's algorithm lies in the tabulation of the (boolean) result of predicate and clause instantiations. In our parsing algorithm, we propose to extend this tabulation so that not only boolean values are stored, but also the successful clause instantiations for the RHS of each instantiated clause. In other terms, we use a 3-dimensional tabulation structure, where entries are of the following form:

$$\Gamma[(i,\vec{\rho})][f_q^{\vec{\rho}}][j] := \ (i_x, \vec{\rho_x})$$

Γ being a table storing the clause identifier and arguments $i_x, \vec{\rho_x}$ corresponding to the instantiation of the jth RHS predicate of the clause i with the qth binding of arguments $\vec{\rho}$.

As a consequence of this extension, after parsing a shared forest can be straightforwardly extracted from the table of clause instantiations. This shared forest is represented by a context-free grammar, following Billot and Lang (1989). See Fig. 5 for an example.

5.2 Using constraints to instantiate predicates

A second extension of Boullier's algorithm concerns the complex task of clause instantiation. During RCG parsing, for each clause instantiation, all possible bindings between the arguments of the LHS predicate and (a substring of) the input string must be computed. The more ranges with free boundaries the arguments of the LHS predicate contains, the more expensive the instantiation is. Boullier (2000) has shown that the time complexity of a clause instantiation is $\mathcal{O}(n^d)$, where n is length of the input string, and d is the arity of the grammar (maximal number of free range boundaries). To deal with this high time complexity, Boullier (2000) proposes to use some predefined specific predicates[7] whose role is to decrease the number of free range boundaries.

In our approach, we propose to encode the clause instantiation task into a *Constraint Satisfaction Problem* (CSP). More precisely, we propose to use constraints over finite sets of integers to represent the constraints affecting the range boundaries. Indeed, these constraints over integers offer a natural way of encoding constraints applied on ranges (*e.g.* linear order).

Let us briefly introduce CSPs. In a CSP, a problem is described using a set of variables, which take their values in a given domain. Constraints are then applied on the values these variables can take in order to narrow their respective domain. Finally, one (or all) solution(s) to the problem are searched for, that is to say some (or all) assignment(s) of values to variables while respecting the constraints are searched for. One particularly interesting subclass of CSPs are those that can be stated in terms of constraints on variables ranging over finite sets

[7] E.g. a *length* predicate is used to limit the length of the subpart of input string covered by a range.

of non-negative integers. For such CSPs, there exist several implementations offering a wide range of constraints (arithmetic, boolean and linear constraints), and efficient solvers, such as the *Gecode* library[8] (Schulte and Tack, 2006).

In this context, the underlying idea of computing range instantiations as a CSP is the following. We use the natural order of integers to represent the linear order of ranges. More precisely, we compute all possible mappings between position indices in the input string (positive integers) and free range boundaries in the arguments of (the LHS predicate of) the clause to instantiate (variables taking their values in $[0..n]$, n being the length of the input sentence). Note that, within a given argument of a predicate to instantiate, a range of type *constant* can be considered as a constraint for the values the preceding and following range boundaries can take, see the example Fig. 6 (x_i are variables ranging over finite sets of integers and c_j are constants such that $c_j = j$).

(LHS-)Predicate instantiation:
$P(aXYdZ) \quad \leftrightarrow \quad P(abcdef)$

Constraint-based interpretation:
$P(x_0 \quad a \quad x_1\ X\ x_2\ Y\ x_3 \quad d \quad x_4\ Z\ x_5) \quad \leftrightarrow$
$P(c_0 \quad a \quad c_1\ b\ c_2\ c\ c_3 \quad d \quad c_4\ e\ c_5\ f\ c_6)$

$$\begin{cases} i \leq j \quad \Rightarrow \quad x_i \leq x_j & \text{(linear order)} \\ x_0 = c_0 \quad x_5 = c_6 & \text{(extern boundaries)} \\ x_1 = c_1 \quad x_3 = c_3 \quad x_4 = c_4 & \text{(anchor constraints)} \end{cases}$$

(here x_2 is the only free range boundary, and can take 3 values, namely c_1, c_2 or c_3)

Figure 6: Constraint-based clause instantiation.

The gain brought by CSP-based techniques remains to be evaluated. So far, it has only been observed empirically between 2 versions of the parser. Nonetheless constraints offer a natural framework for dealing with ranges.[9]

Eventually, note that the extensions introduced in this section do not affect the time complexity of Boullier's algorithm, which is $\mathcal{O}(|G|n^d)$, $|G|$ being the size of the grammar, d its degree, and n the length of the input string.

[8] C.f. `http://www.gecode.org`.

[9] The question of whether feature constraints should be used at this stage or not is discussed in section 6.

6 Retrieving TT-MCTAG derivation structures

As previously mentioned, the result of RCG-parsing is an RCG shared forest. In order to extract from this forest the TT-MCTAG derivation structure (namely the derivation and derived trees), we must first interpret this RCG forest to get the underlying TAG forest, and then expand the latter.

6.1 Interpreting the RCG shared forest

The interpretation of the RCG forest corresponds to performing a traversal of the forest while replacing all *branching* clauses (i.e. clauses whose LHS predicate is labeled by *adj* or *sub*) by the *tree* clause they refer to in the table of clause instantiation. In other terms, each instantiated branching clause is replaced by the tree clause corresponding to its unique RHS-predicate (see Fig. 7).

$$\langle \alpha_{rep}, \emptyset \rangle(es\ der\ Mech\ zu\ rep\ versp) \rightarrow \langle adj, \alpha_{rep}, \epsilon, \{\beta_{acc}\} \rangle(es\ der\ Mech, versp)$$

$$\langle adj, \alpha_{rep}, \epsilon, \{\beta_{acc}\} \rangle(es\ der\ Mech, versp) \rightarrow \langle \beta_{versp.}, \{\beta_{acc}\} \rangle(es\ der\ Mech, versp)$$

$$\langle \beta_{versp}, \{\beta_{acc}\} \rangle(es\ der\ Mech, versp) \rightarrow \langle adj, \beta_{versp}, \epsilon, \{\beta_{acc}, \beta_{nom}\} \rangle(es\ der\ Mech., \epsilon)$$

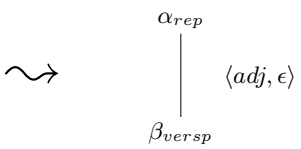


Figure 7: Relation between clause instantiations and TT-MCTAG derivation (using the TT-MCTAG in Fig. 1).

The result of this interpretation of the RCG shared forest is the TT-MCTAG shared forest, i.e. a factorized representation of all TT-MCTAG derivations as a context-free grammar. The extraction of this TT-MCTAG forest is done in a single traversal of the RCG forest (i.e. of the table of clause instantiations) starting from the clause whose LHS predicate is the start predicate. Since the predicate names contain the tree identifiers they refer to, no lookup in the grammar is needed. As a consequence, the time complexity of the extraction of the TT-MCTAG forest is bound by the size of the table of clause instantiations.

Note that (i) we do not expand the alternatives resulting from syntactic ambiguity at this stage,

and (ii) both the RCG and TT-MCTAG derivation forests have been computed without taking the feature structures into account. The motivation is to delay the cost of unification to the final step of expansion of the TT-MCTAG forest. Indeed, the word order constraints encoded in the RCG have possibly rejected many ungrammatical structures for which the cost of feature unification would have been wasted time. It would be interesting to experiment whether we would benefit or not from using feature structures as additional constraints on clause instantiation in practice.

6.2 Expanding the TAG shared forest

Finally, from this TT-MCTAG derivation forest, we can extract all derivation trees, and then compute the corresponding derived trees.

This task amounts to traversing the forest in a top-down-fashion, using the information in the encountered nodes (referring to elementary trees) to gradually assemble derivation trees. Some nodes in the forest encode a syntactic ambiguity (disjunctive node), in which case we make a copy of the current derivation tree and apply one of the alternative options to each of the trees before following each branch through. This behavior is easy to implement using a FIFO queue. A few control mechanisms check for integrity of the derivation trees during the process. We end up with a set of derivation trees in an XML DOM format that can either be displayed directly in the GUI or exported in an XML file.

For reasons of flexibility, we chose to rely on an XML DOM internal representation for all the steps of derived tree building. Indeed, this enables each of the derivation steps to be displayed directly in the GUI. Feature unification also happens at this point, allowing for a graphical illustration of feature clashes in the parse tree in debug mode.

7 Computing semantics

The parsing architecture introduced here has been extended to support the syntax/semantics interface of Gardent and Kallmeyer (2003). The underlying idea of this interface is to associate each tree with flat semantic formulas. The arguments of these formulas are unification variables co-indexed with features labelling the nodes of the syntactic tree. During derivation, trees are combined via adjunction and/or substitution, each triggering the unifications of the feature structures labelling specific nodes. As a result of these unifications, the arguments of the semantic formulas associated with the trees involved in the derivation get unified. In the end, each derivation/derived tree is associated with a flat semantic representation corresponding to the union of the formulas associated with the elementary trees that have been used. An example is given in Fig. 8.

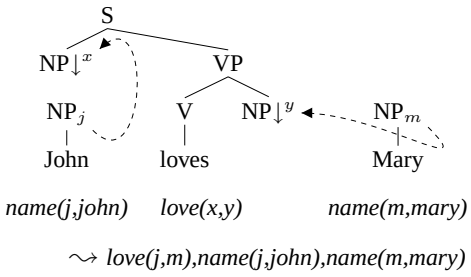


Figure 8: Semantic calculus in Feature-Based TAG.

In our system, the integration of the semantic support has only required 2 extensions, namely (i) the extension of the tree objects to include semantic formulas, and (ii) the extension of the construction of the derived tree so that the semantic formulas are carried until the end and updated with respect to the feature-structure unifications performed.

8 Optimizations

The parsing architecture presented here can host several optimizations. In this section, we present two examples of these. The first one concerns lexical disambiguation, the second one RCG parsing.

Lexical disambiguation becomes a necessity because, for each token of the input sentence, there may be many candidate elementary trees, each of these being used in the RCG conversion, thus leading to a combinatorial explosion for longer sentences.[10] We tackled this problem using the technique introduced in Bonfante et al. (2004). The idea behind their approach is to encode all the possible combinations of elementary trees in an automaton. For this purpose, elementary trees are first reduced to sets of polarity values depending on the *resources* and *needs* they represent (a substitution or foot node refers to a need for a certain category, while a root node corresponds to a re-

[10] Recall that all licensed adjunctions are predicted.

source). For example, an S elementary tree with two places for NP substitution has an NP polarity of -2 and an S polarity of +1.Using this representation, every candidate elementary tree is represented by an edge in an automaton built by scanning the input sentence from left to right. The polarity of a path through the automaton is the sum of all the polarities of the edges encountered on the way. While building this automaton, we determine all the paths with a neutral polarity for every category but the parsed constituent's category (whose polarity is +1). Such a path encodes a set of elementary trees that could contribute to a valid parse. As a consequence, the parser only has to consider for RCG conversion, combinations for a small number of tree sets. This approach makes the search space for both RCG conversion and RCG parsing much more manageable and leads to a significant drop in parsing time for some long sentences.

The second optimization concerns RCG parsing, which can have a high cost in cases where there are many free range boundaries. We can decrease the number of such boundaries by adding a constraint preventing range variables referring to substitution nodes from being bound to ϵ.

9 Conclusion and future work

In this paper, we introduced a parsing environment using RCG as a pivot formalism to parse mildly context-sensitive formalisms such as TT-MCTAG. This environment opens the way to multi-formalism parsing. Furthermore, its modular architecture (RCG conversion, RCG parsing, RCG shared forest interpretation) made it possible to extend the system to perform additional tasks, such as semantic calculus or dependency structure extraction. The system is still being developed, but is already used for the development of a TT-MCTAG for German (Kallmeyer et al., 2008). Future work will include experiments with off-line conversion of TT-MCTAG and generalization of branching clauses to reduce the size of the RCG and thus to improve (RCG) parsing time.

References

Billot, Sylvie and Bernard Lang. 1989. The Structure of Shared Forests in Ambiguous Parsing. In *27th Annual Meeting of the Association for Computational Linguistics*, pages 143–151.

Bonfante, Guillaume, Bruno Guillaume, and Guy Perrier. 2004. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *Proceedings of 20th International Conference on Computational Linguistics (CoLing 2004)*, pages 303–309.

Boullier, Pierre. 1999. On TAG and Multicomponent TAG Parsing. Rapport de Recherche 3668, Institut National de Recherche en Informatique et en Automatique (INRIA).

Boullier, Pierre. 2000. Range concatenation grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT 2000)*, pages 53–64.

Duchier, Denys, Joseph Le Roux, and Yannick Parmentier. 2004. The Metagrammar Compiler: An NLP Application with a Multi-paradigm Architecture. In *Second International Mozart/Oz Conference (MOZ'2004)*.

Gardent, Claire and Laura Kallmeyer. 2003. Semantic Construction in FTAG. In *EACL 2003, 10th Conference of the European Chapter of the Association for Computational Linguistics*, pages 123–130.

Kallmeyer, Laura and Yannick Parmentier. 2008. On the relation between Multicomponent Tree Adjoining Grammars with Tree Tuples (TT-MCTAG) and Range Concatenation Grammars (RCG). In *Proceedings of the 2nd International Conference on Language and Automata Theory and Applications LATA*, pages 277–288.

Kallmeyer, Laura, Timm Lichte, Wolfgang Maier, Yannick Parmentier, and Johannes Dellert. 2008. Developing a TT-MCTAG for German with an RCG-based parser. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008)*. To appear.

Lichte, Timm and Laura Kallmeyer. 2008. Factorizing Complementation in a TT-MCTAG for German. In *Proceedings of the The Ninth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+9)*.

Lichte, Timm. 2007. An MCTAG with tuples for coherent constructions in German. In *Proceedings of the 12th Conference on Formal Grammar*.

Schulte, Christian and Guido Tack. 2006. Views and iterators for generic constraint implementations. In *Recent Advances in Constraints (2005)*, volume 3978 of *Lecture Notes in Artificial Intelligence*, pages 118–132. Springer-Verlag.

Søgaard, Anders, Timm Lichte, and Wolfgang Maier. 2007. The complexity of linguistically motivated extensions of tree-adjoining grammar. In *Recent Advances in Natural Language Processing 2007*.