TuLiPA - Parsing Extensions of TAG with Range Concatenation Grammars

Laura Kallmeyer, Wolfgang Maier, Yannick Parmentier, Johannes Dellert

TuLiPA - Parsing Extensions of TAG with Range Concatenation Grammars

L. Kallmeyer[1], W. Maier[1], Y. Parmentier[2], J. Dellert[1]
[1]*Universität Tübingen, Germany, {lk,wmaier,jdellert}@sfs.uni-tuebingen.de*
[2]*LORIA – Nancy Université, France, parmenti@loria.fr*

Abstract

In this paper we present a parsing framework for extensions of Tree Adjoining Grammars (TAG) called TuLiPA (Tübingen Linguistic Parsing Architecture). In particular, besides TAG, the parser can process Tree-Tuple MCTAG with shared nodes (TT-MCTAG), a TAG-extension that has been proposed to deal with scrambling in free word order languages such as German. The central strategy of the parser is such that the incoming TT-MCTAG (or TAG) is transformed into an equivalent Range Concatenation Grammar (RCG) which, in turn, is then used for parsing. The RCG parser is an incremental Earley-style chart parser. In addition to the syntactic anlysis, TuLiPA computes also an underspecified semantic analysis for grammars that are equipped with 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 (9). Instead of implementing a specific TT-MCTAG parser we follow a more general approach by using Range Concatenation Grammars (RCG) as a pivot formalism. The TT-MCTAG (or TAG) is transformed into a strongly equivalent RCG that is then used for parsing. The motivation for the passage via RCG is that the RCG directly represents the set of derivation trees of the original grammar. Consequently, it abstracts away from traversals of elementary trees without substitutions or adjunctions and the output of the RCG parser can be directly interpreted as the TT-MCTAG (or TAG) derivation forest.

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 freely available under the GPL.[1]

In this paper, we present this parsing architecture focussing on the following aspects: first, we introduce the TT-MCTAG formalism (section 2). Then, we present successively the RCG formalism and the conversion of TT-MCTAG into RCG (section 3). Section 4 explains the parsing algorithm we use for the specific RCGs we obtain from TAG and TT-MCTAG.

2. TT-MCTAG

Tree Adjoining Grammars (6) are a formalism based on tree rewriting. A *Tree Adjoining Grammar* (TAG) is a tuple $G = (V_N, V_T, S, I, A)$ where V_N and V_T are disjoint alphabets of non-terminal and terminal symbols, respectively, $S \in V_N$ is the start symbol, and I and A are finite sets of *initial* and *auxiliary* trees, respectively. Trees in $I \cup A$ are called *elementary* trees. The internal nodes in the elementary trees are labeled with non-terminal symbols, the leaves with non-terminal or terminal symbols. As a special property, each auxiliary tree β has exactly one of its leaf nodes marked as the *foot* node, having the same label as the root. Such a node is denoted by *. Leaves with non-terminal labels that are not foot nodes are called *substitution* nodes.

In a TAG, larger trees can be derived from the elementary trees by subsequent applications of the operations substitution and adjunction. The *substitution* operation replaces a substitution node η with an initial tree having root node with the same label as η. The *adjunction* operation replaces an internal node η in a previously derived tree γ with an auxiliary tree β having root node with the same label as η. The subtree of γ rooted at η is then placed below the foot node of β. Only internal nodes can allow for adjunction, adjunction at leaves is not possible.

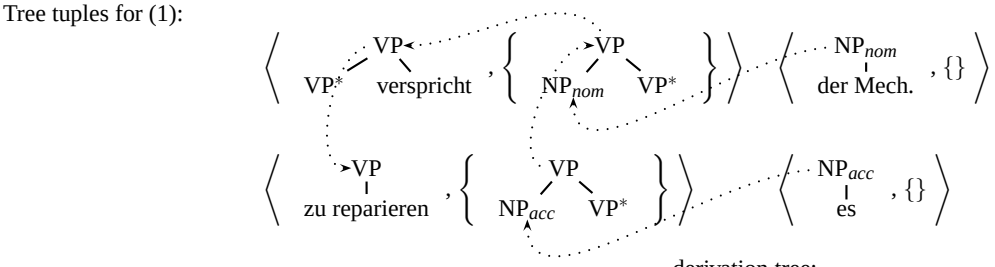
TAG derivations are represented by derivation trees that record the history of how the elementary trees are put together. A *derivation tree* is an unordered tree whose nodes are labeled with elements in $I \cup A$ and whose edges are labeled with Gorn addresses of elementary trees.[2] Each edge in a derivation tree stands for an adjunction or a

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

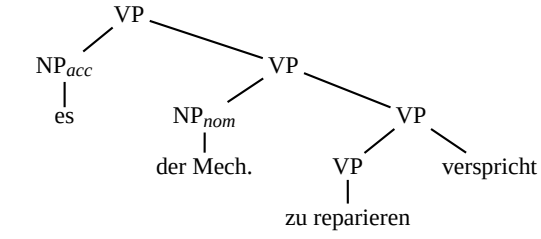
[2] In this convention, the root address is ε and the jth child of a node with address p has address $p \cdot j$.

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

Tree tuples for (1):



derived tree:



derivation tree:

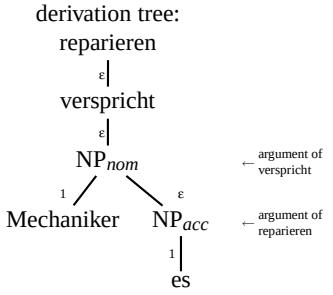


Figure 1: TT-MCTAG analysis of 1

substitution. In the following, we write a derivation tree D as a directed graph $\langle V, E, r\rangle$ where V is the set of nodes, $E \subset V \times V$ the set of arcs and $r \in V$ the root. For every $v \in V$, $Lab(v)$ gives the node label and for every $\langle v_1, v_2\rangle \in E$, $Lab(\langle v_1, v_2\rangle)$ gives the edge label.

TT-MCTAGs (9) 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.

A derivation tree $D = \langle V, E, r\rangle$ in the underlying TAG G_T is licensed in G if and only if the following conditions (MC) and (SN-TTL) are both satisfied.

(MC): For all sets Γ from G and for all γ_1, γ_2 in Γ, we have $|\{v \,|\, v \in V,\, Lab(v) = \gamma_1\}| = |\{v \,|\, v \in V,\, Lab(v) = \gamma_2\}|$.

(SN-TTL): For all argument trees β with head $h(\beta)$ and $n \geq 1$, let $v_1, \ldots, v_n \in V$ be pairwise different nodes with $Lab(v_i) = h(\beta)$, $1 \leq i \leq n$. Then there are pairwise different nodes $u_1, \ldots, u_n \in V$ with $Lab(u_i) = \beta$, $1 \leq i \leq n$. Furthermore, for $1 \leq i \leq n$ either $\langle v_i, u_i\rangle \in E$, or else there are $u_{i,1}, \ldots, u_{i,k}$ with auxiliary tree labels such that $u_i = u_{i,k}$, $\langle v_i, u_{i,1}\rangle \in E$ and, for $1 \leq j \leq k-1$, we have $\langle u_{i,j}, u_{i,j+1}\rangle \in E$ with $Lab(\langle u_{i,j}, u_{i,j+1}\rangle) = \varepsilon$.

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*.

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 called k-TT-MCTAG. Both, TT-MCTAG and the restricted k-TT-MCTAG generate only polynomial langugaes (7; 8).

3. Transforming TT-MCTAG into RCG

The central idea of our parsing strategy is to use RCG (1; 2) as a pivot formalism. RCGs are grammars that rewrite predicates ranging over parts of the input by other predicates. E.g., a clause $S(aXb) \rightarrow S(X)$ signifies that S is true for a part of the input if this part starts with an a, ends with a b, and if, furthermore, S is also true for the part between a and b.

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

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.[3] 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 $L(G) = \{w \mid S(\langle 0, |w| \rangle) \stackrel{*}{\Rightarrow} \varepsilon \text{ wrt } w\}$.

An RCG is *simple* if in all clauses, all righthand side arguments are only single variables and every variable in the lefthand side occurs exactly once in the righthand side and vice versa. The RCGs we are dealing with are all simple. Simple RCGs are equivalent to linear context-free rewriting systems (LCFRS).

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 (1). 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.

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.

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

Figure 2: Some clauses of the RCG correponding to the TT-MCTAG in Fig. 1

As an example see Fig. 2. 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 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.

4. 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 since the cost of the conversion depends heavily on the size of the grammar (all licensed adjunctions have to be computed while taking into account the state of the list of pending arguments).[4]

[3] 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] We do not have a proof of complexity of the conversion algorithm yet, but we conject 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.

The RCGs one obtains from transforming a *k*-TT-MCTAG are always ordered simple RCGs. A simple RCG is *ordered* if the order of variables is the same in the lhs and rhs predicates of a clause. In particular, this means that for every clause instantiation, the order of arguments of a predicate is the same as the order of the correponding ranges in the input string.

The algorithm used in our parser is a modification of (3) and is very close to the strategy adopted in (10). It is an incremental Earley-style chart parser.

The general idea is as follows: We process the arguments of lefthand sides of clauses incrementally, starting from an *S*-clause. Whenever we reach a variable, we move into the clause of the correponding righthand side predicate. The first time, this is done by predicting the righthand side predicate and starting the traversal of its first argument. When moving into the clause of a righthand side predicate that was already started, we resume it. Whenever we reach the end of an argument, we suspend this clause and move into the parent clause that has called the current one. In addition, we treat the case where we reach the end of the last argument and move into the parent as a special case. Here, we first convert the item into a passive one and then complete the parent item with this passive item. This allows for some additional factorization. We have implemented this strategy as a chart parser.

As a result of the parser, we obtain a RCG derivation forest that can be directly interpreted as the underlying TAG derivation forest since the RCG predicates represent the elementary trees used in the derivation and the substitutions and adjunctions performed.

The parsing architecture introduced here has been extended to support the syntax/semantics interface of Gardent and Kallmeyer (4). 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.

References

[1] Pierre Boullier. On TAG Parsing. In *TALN 99, 6^e conférence annuelle sur le Traitement Automatique des Langues Naturelles*, pages 75–84, Cargèse, Corse, July 1999.

[2] Pierre Boullier. Range Concatenation Grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies (IWPT2000)*, pages 53–64, Trento, Italy, February 2000.

[3] Håkan Burden and Peter Ljunglöf. Parsing linear context-free rewriting systems. In *IWPT'05, 9th International Workshop on Parsing Technologies*, Vancouver, Canada, October 2005.

[4] Claire Gardent and Laura Kallmeyer. Semantic Construction in FTAG. In *Proceedings of EACL 2003*, pages 123–130, Budapest, 2003.

[5] Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. Tree Adjunct Grammars. *Journal of Computer and System Science*, 10:136–163, 1975.

[6] Aravind K. Joshi and Yves Schabes. Tree-Adjoning Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69–123. Springer, Berlin, 1997.

[7] Laura Kallmeyer and Yannick Parmentier. On the relation between Multicomponent Tree Adjoining Grammars with Tree Tuples (TT-MCTAG) and Range Concatenation Grammars (RCG). In Carlos Martín-Vide, Friedrich Otto, and Henning Fernaus, editors, *Language and Automata Theory and Applications. Second International Conference, LATA 2008*, number 5196 in Lecture Notes in Computer Science, pages 263–274. Springer-Verlag, Heidelberg Berlin, 2008.

[8] Laura Kallmeyer and Giorgio Satta. A Polynomial-Time Parsing Algorithm for TT-MCTAG. In *Proceedings of ACL*, Singapore, 2009.

[9] Timm Lichte. An MCTAG with Tuples for Coherent Constructions in German. In *Proceedings of the 12th Conference on Formal Grammar 2007*, Dublin, Ireland, 2007.

[10] Éric Villemonte de La Clergerie. Parsing mildly context-sensitive languages with thread automata. In *Proc. of COLING'02*, August 2002.