Developing a TT-MCTAG for German with an RCG-based Parser

Laura Kallmeyer, Timm Lichte, Wolfgang Maier, Yannick Parmentier†, Johannes Dellert

SFB 441, †SfS-CL, University of Tübingen
Nauklerstr. 35, 72074 Tübingen, Germany
lk@sfs.uni-tuebingen.de, timm.lichte@uni-tuebingen.de, {wmaier,parmenti,jdellert}@sfs.uni-tuebingen.de

Abstract

Developing linguistic resources, in particular grammars, is known to be a complex task in itself, because of (amongst others) redundancy and consistency issues. Furthermore some languages can reveal themselves hard to describe because of specific characteristics, e.g. the free word order in German. In this context, we present (i) a framework allowing to describe tree-based grammars, and (ii) an actual fragment of a core multicomponent tree-adjoining grammar with tree tuples (TT-MCTAG) for German developed using this framework. This framework combines a metagrammar compiler and a parser based on range concatenation grammar (RCG) to respectively check the consistency and the correction of the grammar. The German grammar being developed within this framework already deals with a wide range of scrambling and extraction phenomena.

1. Introduction

Among existing grammatical formalisms, tree-adjoining grammar (TAG) (Joshi and Schabes, 1997) distinguishes itself by its expressivity, which lies beyond context-free grammar, and its nice computational properties (e.g. polynomial parsing time complexity). It has been used for implementing a large-scale grammar of English (XTAG Research Group, 2001), as well as grammars for French, Chinese and Korean. However, the creation of a grammar resource for German within a TAG framework poses a challenge, mainly due to frequently occurring scrambling phenomena in German.

The purpose of this paper is to present an architecture for the development of a German TAG-based grammar, specifically taking care of the difficulties introduced by that language. The architecture relies on a TAG-based grammar formalism and includes a metagrammar compiler and a parsing system to respectively ensure the grammar consistency, and the grammar correctness.

The TAG-based formalism chosen for describing German is multicomponent tree-adjoining grammar with tree tuples (TT-MCTAG) (Lichte, 2007). The metagrammar environment used is XMG (Duchier et al., 2004), and the parser for TT-MCTAG the TuLiPA system[1], whose core component is a parser for range concatenation grammar (RCG) (Boullier, 2000).

The paper is structured as follows. In section 2., we introduce the TT-MCTAG formalism highlighting its adequacy for describing German. Then, in section 3. we briefly introduce the XMG metagrammar environment and its use for developing electronic tree-based grammars. In section 4., we present the ideas underlying the TuLiPA parsing environment, namely the use of RCG as a pivot formalism, opening the way to parsing mildly context-sensitive formalisms in general and TT-MCTAG in particular. Then, in section 5., we present the German grammar under development and give some prospects. Finally (section 6.), we compare our approach with existing work.

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

2. TT-MCTAG: A Grammar Formalism for German

As underlying grammar formalism for our German grammar, we make use of an extension of tree-adjoining grammar (TAG) (Joshi and Schabes, 1997), a tree-rewriting formalism. Roughly put, a TAG consists of a set of elementary trees that can be combined using two operations: substitution (replacement of frontier nodes) and adjunction (replacement of inner nodes). Besides the derived tree, TAG generates a second structure, the derivation tree, that encodes the way the derived tree was obtained. More specifically, the derivation tree contains one node for each elementary tree used during the derivation and one edge for each adjunction/substitution.

While a rather pure TAG could be used within the XTAG system for English, German, due to its rather free word order, is known to resist a satisfying analysis based on the expressive power of TAG alone. To overcome this, extensions of TAG with multiple components (MCTAG) have been proposed, among which multicomponent TAG with tree tuples (TT-MCTAG) (Lichte, 2007) has been chosen for TuLiPA.

In TT-MCTAG, elementary structures are made of tuples of the form $\langle \alpha, \{\beta_1, ..., \beta_n\}\rangle$, where $\alpha, \beta_1, ..., \beta_n$ are elementary trees in terms of TAG. During derivation, the β-trees have to attach to the α-tree, either directly or indirectly via node sharing. Roughly speaking, node sharing terms an extended locality that allows β-trees to also adjoin to the root of trees that either adjoin to α themselves, or that are again in a node sharing relation to α. In other words, an argument must be linked to a tree adjoining to its head by a chain of root adjunctions.

We further restrict TT-MCTAG, 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. TT-MCTAG in general are NP-complete (Søgaard et al., 2007) while k-TT-MCTAG are mildly context-sensitive (Kallmeyer and Parmentier, 2008).

As shown in Lichte (2007), TT-MCTAG is suitable for analyzing free word order phenomena in German, such as

scrambling and coherent constructions. The linguistic understanding of a tuple is that of a head (the α-tree) and its subcategorization frame (the β-trees). As an example see the tuple for *vergisst* ('forgets') in Figure 1, which shows the derivation tree for the word order in (1)a.

(1) a. *dass Peter ihn heute vergisst*
b. *dass ihn Peter heute vergisst*
c. *dass ihn heute Peter vergisst*
d. *dass heute ihn Peter vergisst*
e. ...
('that Peter forgets him/it today')

It indicates that, starting from the *vergisst* head tree, the adverbial *heute* ('today') first adjoins to the root address (Gorn address 0), then the accusative argument tree adjoins to the root of the adverbial, and then the nominative argument tree to the root of the accusative tree. Each argument is linked by a chain of root adjunctions to *heute* which adjoins to the head.
While TT-MCTAG provides satisfying descriptive means to cope with free word order phenomena in German, it is not obvious how to implement a TT-MCTAG of a wide lexical coverage. We use the means of factorization provided by the metagrammar environment XMG, presented in the next section.

3. Using a Metagrammar Formalism for Grammar Implementation

The development of large linguistic resources, especially grammars, is known to be a complex task (Erbach and Uszkoreit, 1990). On top of being time-consuming, grammar engineering has to deal with consistency and maintenance issues. Indeed, when the grammar reaches a certain size in terms of number of rules, it is hard (if not impossible) to guarantee the consistency between these. Furthermore, a modification of some information included in the rules may lead to the modification of a substantial part of the grammar. To avoid these problems, several techniques have been proposed, including the *metagrammar* approach. The idea underlying metagrammars is that the rules composing a grammar follow some linguistic invariants[2], and that one should be able to automatically produce the grammar from a linguistic description of these invariants.
For our grammar implementation, we are using the *eXtensible MetaGrammar* (XMG) environment, which offers an expressive language for describing tree-based grammars (namely TAG, MCTAG and Interaction Grammars), along with a Warren Abstract Machine-based implementation (Duchier et al., 2004). XMG allows to factorize the description of single trees in the grammar into different pieces, each describing a tree fragment that can be used in different contexts to combine with other fragments to form actual elementary trees. This compact way to describe a tree-based grammar allows to avoid redundancies and thereby helps to guarantee consistency.
The XMG language provides the metagrammar designer with two levels of description[3]:

- a tree description language allowing one to define tree fragments using the logic described in Rogers and Shanker (1992),
- a combination language allowing to combine tree fragments either conjunctively or disjunctively (the latter making it possible to define alternative syntactic structures, *e.g.* active / passive).

As an illustration of a metagrammatical description, consider the tree fragments associated respectively with a canonical subject, a relativized subject, an active verbal morphology and a canonical subject, as depicted in Figure 2. These fragments are combined via the following rule:

$$Transitive \rightarrow (SubjectCan \vee SubjectRel) \wedge Active \wedge ObjectCan$$

The result of this combination is the production of two trees of a TAG, namely the tree associated with transitive verbs having a canonical object and either a canonical or a relativized subject.
From this example, it is worth noticing that the metagrammar does not actually describe lexicalized trees, but *tree schemata* as introduced in the XTAG grammar (XTAG Research Group, 2001). A tree schema is a tree where one leaf node is marked with $\diamond$ and referred to as the anchor node, i.e. the node that will receive the lexical item. Thus, tree schemata are abstractions over lexicalized trees. They correspond to tree structures that are common to words sharing specific properties such as valency and syntactic category. Furthermore, the trees are gathered into *families* (for verbs, a family corresponds to a subcategorization frame). This tree gathering makes it easier to define the associations between words and tree schemata. These associations are defined within a 2-layer lexicon. First, one defines a morphological lexicon mapping words with lemmas and morphological features (e.g. gender). Secondly, one defines a syntactic lexicon mapping lemmas with tree families and syntactic features (e.g. reflexivity of a verb).
The parser is thus given a grammar made of tree schemata, plus this 2-layer lexicon as an input, and realizes the tree anchoring prior to parsing. The conceptual interest of handling tree schemata instead of lexicalized trees lies in the abstraction theses schemata provide, and its practical interest lies in the reduction of redundancy. An example of entries of morphological and syntactic lexica are given in section 5.
In the context of the implementation of an MCTAG for German, XMG has been extended to describe not only trees but also sets of trees (Parmentier et al., 2007). This extension lies in the interpretation of node variables that do not have any ancestor (the so-called local roots). When dealing with descriptions of trees, one has to make sure that the models are actual trees, i.e. that there is a unique root. Thus, if the description contains several local roots, all possible node identifications are computed in order to obtain a tree

[2] E.g. a transitive verb uses a subject and an object in an active form.

[3] For more details about the XMG language and its use for grammar development, see Crabbé (2005a).

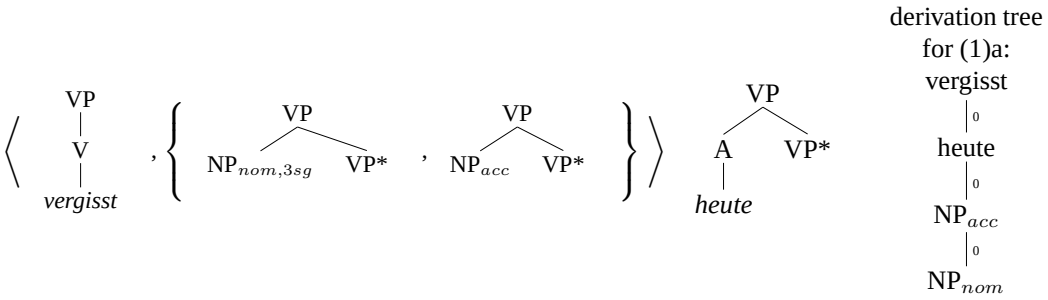


Figure 1: Tree tuples for (1)a

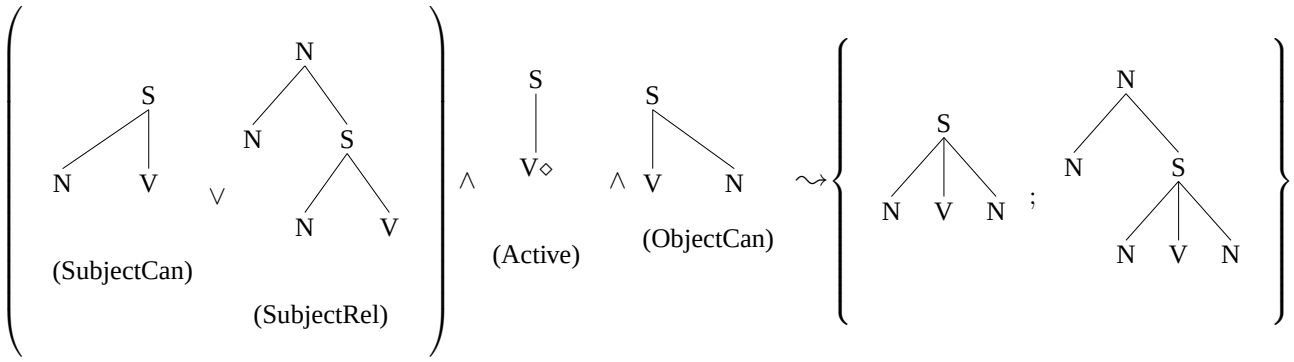


Figure 2: Combination of tree fragments within a metagrammatical description.

model. When dealing with descriptions of sets of trees, this unique root constraint has no longer to be enforced. In other terms, if a description formula contains two node variables for which there is no ancestor and provided this formula has a model, this model is a set made of two trees.
Note that the notion of family still exists in this case, but now refers to sets of sets of trees, where a given set contains only one tree with an anchor.

4. Parsing of TT-MCTAG

In the previous section, we introduced the XMG environment allowing the linguist to define a factorized description of a grammar capturing its redundancy. In this section, we present a parsing architecture supporting TT-MCTAG, which makes it possible to check the correction and coverage of the grammar.
Note that prior to this work, there was no parser available for TT-MCTAG. We thus designed and implemented a parser for this formalism, using RCG as a pivot formalism. In this context, parsing is done as follows. First a subgrammar is selected according to the input sentence. This subgrammar is then converted into an RCG, used to parse the input sentence. The result of parsing is an RCG derivation forest[4], that is interpreted to extract a TT-MCTAG derivation forest. The derivation / derived trees are finally extracted from the TT-MCTAG forest and additional processings are performed (e.g. extraction of dependency views and computation of semantic representations).
Section 4.1. introduces the specific TT-MCTAG to RCG conversion algorithm and section 4.2. presents the architecture of the TuLiPA parser relying on this algorithm.

[4]A derivation forest is a shared representation of all parses for a given sentence.

4.1. Using RCG as a pivot formalism

In order to parse TT-MCTAG, we first construct an equivalent simple range concatenation grammar and then use this RCG for parsing. The advantage of this approach is that the architecture becomes more modular and parts of it (in particular, the RCG parser) can be reused for parsing other mildly context-sensitive formalisms.
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 $A_0(x_{01}, \ldots, x_{0a_0}) \rightarrow A_1(x_{11}, \ldots, x_{1a_1}) \ldots A_n(x_{n1}, \ldots, x_{na_n})$ with $n \geq 1$ and $A_i \in N, x_{ij} \in (T \cup V)^*$ 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.[5] 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[6] $L(G) = \{w \mid S(\langle 0, |w|\rangle) \stackrel{*}{\Rightarrow} \epsilon \text{ wrt } w\}$. As an illustration of RCG derivation, see Figure 3.
The construction of an RCG for a given k-TT-MCTAG is similar to the RCG construction for TAG (Boullier, 1999). The general idea of the latter is as follows: the RCG con-

[5]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 $t_{i+1} \ldots t_j$.

[6]$\Rightarrow$ refers to the derivation operation, and $\stackrel{*}{\Rightarrow}$ to its reflexive transitive closure.

RCG: $G = \langle \{S, A, B\}, \{a, b\}, \{X, Y, Z\}, S, P \rangle$
$S(X\,Y\,Z) \rightarrow A(X, Z)\,B(Y)$, $A(a\,X, a\,Y) \rightarrow A(X, Y)$,
$B(b\,X) \rightarrow B(X)$, $A(\epsilon, \epsilon) \rightarrow \epsilon$, $B(\epsilon) \rightarrow \epsilon$.
Input: $w = aabaa$.
Derivation:

$S(XYZ) \rightarrow A(X,Z)B(Y)$

$\langle 0,2\rangle\langle 2,3\rangle\langle 3,5\rangle \;\; \langle 0,2\rangle\langle 3,5\rangle\langle 2,3\rangle$
$aa \quad b \quad aa \quad\quad aa \quad aa \quad b$

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 $B(\epsilon) \rightarrow \epsilon$

$\langle 2,3\rangle\langle 3,3\rangle\langle 3,3\rangle$
$b \quad \epsilon \quad \epsilon$

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

$A(aXaY) \rightarrow A(X,Y)$

$\langle 0,1\rangle\langle 1,2\rangle\langle 3,4\rangle\langle 4,5\rangle\langle 1,2\rangle\langle 4,5\rangle$
$a \quad a \quad a \quad a \quad a \quad a$

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 $A(\epsilon, \epsilon) \rightarrow \epsilon$

$\langle 1,2\rangle\langle 2,2\rangle\langle 4,5\rangle\langle 5,5\rangle\langle 2,2\rangle\langle 5,5\rangle$
$a \quad \epsilon \quad a \quad \epsilon \quad \epsilon \quad \epsilon$

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

Figure 3: Sample RCG

tains predicates $\langle\alpha\rangle(X)$ and $\langle\beta\rangle(L, R)$ for initial and auxiliary trees respectively. X covers the yield of α and all trees added to α, while L and R cover those parts of the yield of β (including all trees added to β) that are to the left and the right of the foot node of β. The clauses in the RCG reduce the argument(s) of these predicates by identifying those parts that come from the elementary tree α/β itself and those parts that come from one of the elementary trees added by substitution or adjunction.

Each TT-MCTAG derivation is a TAG derivation since it composes elementary trees using adjunction and substitution. In the RCG we construct for a TT-MCTAG, there are predicates $\langle\gamma\rangle$ for the elementary trees (not the tree sets). The yield of $\langle\gamma\rangle$ 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 order to keep track of the higher arguments still waiting for adjunction, we enrich the predicate names with the "list of pending arguments" (LPA).

This leads to an RCG of arity 2 with complex predicate names. In order to keep the number of necessary predicates finite, the limit k is crucial, since k gives us the maximal LPA length.

In addition to the $\langle\gamma\ldots\rangle$ predicates we also use branching predicates $\langle adj\ldots\rangle$ and $\langle sub\ldots\rangle$ that take care of the adjunctions/substitutions possible at a given node.

As an example see the TT-MCTAG tuples for (2) and some of the equivalent RCG clauses in Figure 4.

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

The first clause states that, starting from the initial tree α_{rep} for *reparieren*, we can decompose its yield into the left part of a root-adjoining tree, followed by *zu reparieren*, followed by the right part of a root adjoining tree. The predicate $\langle adj, \alpha_{rep}, 0, \{\beta_{acc}\}\rangle$ is reponsable for computing the possible root (position 0) adjunctions in α_{rep}. β_{acc} is added to the set representing the LPA since we have to keep track of the fact that it needs to be adjoined at some point. The clauses for the $\langle adj, \alpha_{rep}, 0, \{\beta_{acc}\}\rangle$ predicate tell us that we can either adjoin β_{acc} (while removing it from the LPA) or we can adjoin a new head tree, β_v of *verspricht* while keeping β_{acc} on the LPA.

Note that with this construction, the grouping into sets gets lost. However, in our parser, the transformation is done only for the small set of tree sets selected for the input sentence. Additionally, whenever an input symbol occurs twice, we use two different occurrences of the same set having different identifiers. This way, we can identify trees belonging to the same set by searching for the same tree set identifier.

The conversion process is described in detail in Kallmeyer and Parmentier (2008).

4.2. TuLiPA: An RCG-based parser

The TuLiPA system includes the following components:

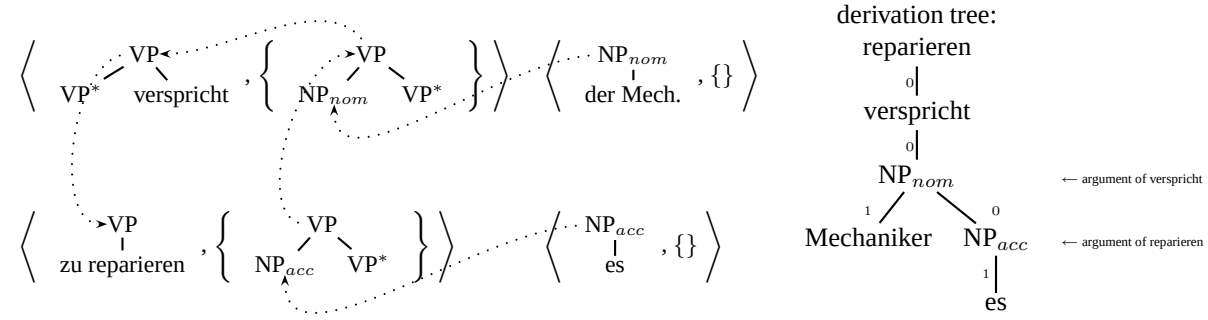
- a TT-MCTAG-to-RCG converter (as introduced in the previous section),
- an RCG parser producing an RCG derivation forest,
- an RCG derivation forest interpreter producing a TT-MCTAG derivation forest,
- a TT-MCTAG derivation forest processor building derivation and derived trees, extracting dependency views and computing semantic representations.

RCG parsing The RCG parsing is done using an algorithm derived from that of Boullier (2000). The recognizer corresponds to the original algorithm, while for parsing, several additions have been made to ensure the correctness of the output[7]. In TuLiPA's RCG parser, the clause instantiations occurring during parsing are tabulated so that once ϵ is derived (i.e. successful parse), an RCG derivation forest can be straightforwardly extracted from the table of clause instantiations (see Figure 5).

RCG derivation forest interpreter After the completion of RCG parsing, the TT-MCTAG derivation forest is extracted from the successfully instantiated clauses. Remember that the RCG clauses represent the adjunctions and substitutions that may occur on a node during the TAG derivation. To extract the forest, we look at the left and right-hand-sides of each clause. These give respectively the site and inserted tree, see Figure 6 for an example of clause interpretation.

Note that both the RCG and TT-MCTAG derivation forests correspond to AND-OR graphs. This follows the results of Billot and Lang (1989).

[7]The original algorithm includes in the output non-valid subderivations.



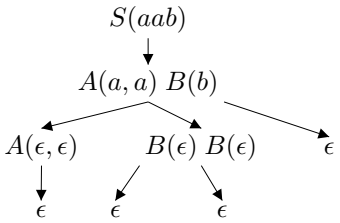
Some clauses of the equivalent RCG:
$\langle \alpha_{rep}, \emptyset \rangle (L\ zu\ reparieren\ R) \rightarrow \langle adj, \alpha_{rep}, 0, \{\beta_{acc}\} \rangle (L, R)$
$\langle adj, \alpha_{rep}, 0, \{\beta_{acc}\} \rangle (L, R) \rightarrow \langle \beta_{acc}, \emptyset \rangle (L, R) \mid \langle \beta_v, \{\beta_{acc}\} \rangle (L, R)$
$\langle \beta_{acc}, \emptyset \rangle (L\ X, R) \rightarrow \langle adj, \beta_{acc}, 0, \emptyset \rangle (L, R) \langle sub, \beta_{acc}, 1 \rangle (X)$
$\langle sub, \beta_{acc}, 1 \rangle (X) \rightarrow \langle \alpha_{es}, \emptyset \rangle (X) \qquad \langle \alpha_{es}, \emptyset \rangle (es) \rightarrow \epsilon$
$\langle \beta_v, \{\beta_{acc}\} \rangle (L, verspricht\ R) \rightarrow \langle adj, \beta_v, 0, \{\beta_{nom}, \beta_{acc}\} \rangle (L, R)$
...

Figure 4: TT-MCTAG analysis of (2) and parts of the equivalent RCG

RCG Grammar:

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	$B(\epsilon)$	$\rightarrow$	ϵ
C_4	$B(b)$	$\rightarrow$	ϵ
C_5	$A(\epsilon,\epsilon)$	$\rightarrow$	ϵ

RCG Derivation wrt aab**:**



RCG derivation forest:

$$C_0(X := a, Y := a, Z := b) \rightarrow (\ C_1(X := \epsilon, Y := \epsilon) \vee C_2(X := \epsilon, Y := \epsilon)\ ) \wedge C_4$$
$$C_1(X := \epsilon, Y := \epsilon) \rightarrow C_5$$
$$C_2(X := \epsilon, Y := \epsilon) \rightarrow C_3 \wedge C_3$$

Figure 5: RCG derivation and corresponding derivation forest.

$$\langle \alpha_{rep}, \emptyset \rangle (es\ der\ Mech\ zu\ rep\ versp) \rightarrow$$
$$\langle adj, \alpha_{rep}, 0, \{\beta_{acc}\} \rangle (es\ der\ Mech, versp)$$
$$\langle adj, \alpha_{rep}, 0, \{\beta_{acc}\} \rangle (es\ der\ Mech, versp) \rightarrow$$
$$\langle \beta_{versp.}, \{\beta_{acc}\} \rangle (es\ der\ Mech, versp)$$

$$\rightsquigarrow \quad \begin{array}{c} \alpha_{rep} \\ | \\ \langle adj, 0 \rangle \\ | \\ \beta_{versp} \end{array}$$

Figure 6: Interpretation of a clause instantiation in terms of TT-MCTAG derivation (using the TT-MCTAG in Fig. 4).

TT-MCTAG derivation forest processor Eventually, several post-processing steps are applied to the TT-MCTAG derivation forest. The first of these expands the forest in order to compute the derivation and derived trees. Recall that the forest contains all derivation steps (that is, the forest corresponds to a directed acyclic graph). It is thus possible to extract all derivation trees it encodes by traversing the graph. For each extracted derivation tree, we can compute the corresponding derived tree.

The other post-processing steps of the derivation forest[8] correspond to (i) the extraction of dependency views, and (ii) the computation of semantic representations.

Concerning (i), the idea is to interpret the relation between the head and the arguments of each tuple used within a parse in terms of dependencies[9].

Concerning (ii), TuLiPA has been extended to support semantically annotated TT-MCTAG. The syntax / semantics interface adopted is the one of Gardent and Kallmeyer (2003). In this interface, each syntactic tree is associated with a predicative semantic formula, whose arguments are unification variables. These variables are co-indexed with features labelling specific nodes of the syntactic tree. The unifications occurring during derivation bind the semantic arguments referring to the same entities together, thus

[8] More precisely of each extracted derivation tree.

[9] These dependencies are displayed using the DTool processor, with courtesy of Marco Kuhlmann.

realizing the semantic composition. To integrate this semantic processing within TuLiPA, the only modifications needed were (a) the extension of the internal representation of tree sets to include semantic formulas and (b) the update of these formulas while the derived tree are computed.

Availability of the system TuLiPA is written in Java and released at `http://www.sfb441.uni-tuebingen.de/emmy/tulipa` under a GNU General Public License. TuLiPA's main features include:

- a relatively easy installation procedure (the only requirement is the GecodeJ[10] library),
- two interfaces, an intuitive graphical one, and a text-based one,
- an output graphical interface displaying the result of parsing (namely derivation tree, derived tree and flat semantic representations),
- an XML export of the parsing result, allowing for the integration of TuLiPA within an NLP processing chain,
- the possibility to use an external Part-Of-Speech tagger (namely the TreeTagger developed at the University of Stuttgart),
- the possibility to parse either tree-based grammars, or RCGs directly.

5. The German TT-MCTAG: Current State and Prospects

After having sketched both the grammar formalism and the implementation framework, i.e. rather formal aspects, in this section, we are turning to a presentation of the linguistic ressources we are aiming at. This includes two dimensions, namely (1) the extend of the syntactic coverage of the grammar and (2) the strategies to expand the lexicon. As this subpart contains still ongoing work, many details inevitably are of a prospective and preliminary nature. Concerning semantic coverage, the addition of the syntax/semantics interface of Gardent and Kallmeyer (2003) to the tree schemata has just started.

Syntactic coverage In our implementation framework, covering syntactic phenomena means specifying appropriate tree (tuple) families lacking an anchor, i.e. a lexical terminal. The respective specification is carried out using the XMG description language and the XMG compiler (see section 3.).

So far, we have implemented (amongst others) free word order phenomena such as scrambling, coherent constructions and verbal clustering. Furthermore, so-called extraction phenomena found in relative clauses, wh-questions and bridging constructions are accounted for. Thanks to TT-MCTAG, factorization from the many possible word orders in German syntax is already supported by the grammar formalism, such that one tree tuple can license several derived structures. To give an example, one tree tuple for each finite verb is sufficient to license most of the extraction phenomena, and scrambling therein. Having this highly factorized description due to the metagrammar and the grammar formalism is not an end in itself, however. Eventually, the concise, but surface-abstracted account for complementation relieves the lexicon.

While the above mentioned phenomena are satisfactorily covered, there turned out to be (well-known) hard nuts such as partial fronting, extraposition, and ellipsis. We hope to include them as well in the future, at least to a certain degree.

For evaluation of the syntactic coverage of the grammar, we will make use of the TSNLP-testsuite for German (Lehmann et al., 1996). Therefore, a handcrafted lexicon is presently being set up.

Morphological specification:

vergisst	vergessen	[pos=v,num=sg,per=3]

Lemma specification:

*ENTRY: vergessen
*CAT: v
*SEM: BinaryRel[pred=vergessen]
*ACC: 1
*FAM: Vnp2
*FILTERS: []
*EX:
*EQUATIONS:
NParg1 → cas = nom
NParg2 → cas = acc
*COANCHORS:

Figure 7: Morphological and lemma specification of *vergisst*.

Lexical coverage As mentioned in section 3., the lexicon is 2-layered: a morphological lexicon maps an (inflected) token to some lemma form, while preserving morphological information in a feature structure; a lemma lexicon basically maps the lemmata onto tree tuple families, while also performing case assignment. Fig. 7 contains the respective entries for the finite verb *vergisst* ('forgets'). The morphological entry consists of the following columns, separated by tabulation: token × lemma × morphological features. The lemma specification is build up by a fixed set of features, of which the most important ones are `*ENTRY` (= the lemma), `*CAT` (= the POS-tag), and `*FAM` (= the tree family as predefined in XMG). Case assignment can be specified in `*EQUATIONS`, where in this example `NParg1` is a defined label of some node in the tree tuples of the specified tree tuple family.

Since German shows rich inflection, this approach helps to reduce redundancies in the lexicon. In sum the lexicon incorporates at least the following information for each token-lemma combination: lemmatization, morphological features, valency information, and the POS-tag. We also started to feed the lexicon with semantic information (see `*SEM`).

When creating the lexicon, we first manually record lexically rather closed classes such as pronouns, determiners

[10] `http://www.gecode.org/gecodej`

and prepositions for the sake of correctness and completeness. Since much of the inflectional information is contributed by these functional items in German, this is advisable. However, in order to obtain a rich lexicon also with respect to lexically rather open classes such as nouns, adjectives and verbs, already existing resources need to be combined without much modification. The combination of several ressources is necessary, since no ressource is known to us, that offers all the mentioned lexical information:

- morphological information: NEGRA corpus[11]
- lemmatization and POS-tags: TüPP-D/Z[12]
- valency information: subcategorization frames from Schulte im Walde (2002)

6. Comparable Work

The German TT-MCTAG for the TuLiPA system is currently under development. However, the already existing fragment reveals interesting differences to other grammar projects from the TAG-realm.

XTAG The XTAG grammar for English (XTAG Research Group, 2001) uses plain TAG, enhanced with feature structures. Covering a wide range of syntactic structures and bringing along a rich lexicon, it is a well developed exemplar. Its implementation framework, however, is different from our one, not only because of the unlike grammar formalisms. XTAG does not use a metagrammar such as XMG for the description of its tree families. It does, however, possess a 2-layered lexicon (*Morph Database* and *Syntactic Database*), that divides the task similarly. Part of the evaluation is also performed on the (English) TSNLP-data. Remains to say, that XTAG does not offer a semantic dimension.
While we cannot compare performance and coverage yet, there are some differences with respect to linguistic design, that are worth mentioning.

1. In contrast to XTAG, we completely omit empty categories (e.g. traces, PRO) in syntactic description. This follows from rejecting a base word order for German, as well as dealing with argument raising and control only in the semantics.

2. Prepositional complements are not encoded via coanchoring as in XTAG. Instead, prepositions are seen as nouns that display a special case feature, that the verb subcategorizes for.

3. In XTAG, extraction phenomena such as relative clauses and wh-questions require separate elementary structures. This is not the case in TuLiPA thanks to a higher degree of factorization in our tree sets compared to standard TAG.

[11] `http://www.coli.uni-saarland.de/projects/sfb378/negra-corpus/negra-corpus.html`

[12] `http://www.sfs.uni-tuebingen.de/en_tuepp.shtml`

French TAG In his thesis, Benoit Crabbé (Crabbé, 2005b) designed a French TAG of considerable syntactic coverage using the XMG framwork. As in XTAG, a plain TAG with feature structures was used. The grammar was also successfully evaluated against the TSNLP-dataset for which a suitable lexicon was implemented. Other than in XTAG, there is also a semantic extension available (Gardent, 2006).

7. Conclusion

In this paper, we have presented an architecture for implementing a TAG-based grammar fragment for German. Because of the limitation of standard TAG, we had to choose a multicomponent TAG variant, TT-MCTAG, as grammar formalism. TT-MCTAG has been argued to be suitable for modelling free word order phenomena. With some additional constraint, the formalism is mildly context-sensitive.
For grammar development, we use a meta-formalism, XMG, that allows a compact grammar representation and enables us to avoid the redundancies that are one of the major problems of grammar implementation. We had to extend XMG to allow for the description of multicomponent TAG.
In order to do parsing with our variant of TAG, we had to develop a new parser. Here we chose an RCG-based parsing architecture that is intended to cover different mildly context-sensitive formalisms (so far we can parse TAG, TT-MCTAG and of course RCG). The central idea of our parser is that RCG is used as a pivot formalism, i.e., some input grammar is transformed into a RCG which is then used for parsing. Søgaard (2007) has described transformations into RCG for a range of other formalisms. These might be included into the TuLiPA parsing system in the future.
For TAG and TT-MCTAG, our parser computes both, syntax and semantics. Here we follow the approach of Gardent and Kallmeyer (2003) that links flat semantic representations to the elementary trees while exploiting the TAG feature unifications for semantic computation.
Finally, we presented a German TT-MCTAG-based grammar, that is currently being developed and lexically expanded using the described implementation framework. By now, it already covers a wide range of challenging syntactic phenomena such as free word order and extraction. In the near future, both the grammar and the lexicon are going to be published under GPL.
We want to emphasize that TuLiPA is not only a parsing architecture useful in the context of TT-MCTAG grammar implementation but it is also one of the first TAG parsers including syntax and semantics. Due to its graphical interface and the various output formats, it helps to increase the transparency and modularity of TAG grammar development. Thanks to these features, it is already at the current state a very useful tool for understanding fundamental TAG principles and therefore can in particularly be successfully used for teaching purposes.

8. References

Sylvie Billot 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, Vancouver, Canada.

Pierre Boullier. 1999. On TAG and Multicomponent TAG parsing. Rapport de recherche, Institut National de Recherche en Informatique et en Automatique.

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

Benoit Crabbé. 2005a. Grammatical development with XMG. In *Proceedings of the 5th International Conference on the Logical Aspects of Computational Linguistics (LACL05)*, Bordeaux, France.

Benoit Crabbé. 2005b. *Représentation informatique de grammaires fortement lexicalisées : Application à la grammaire d'arbres adjoints*. Ph.D. thesis, Université Nancy 2, Nancy, France.

Denys Duchier, 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)*, Charleroi, Belgium.

Gregor Erbach and Hans Uszkoreit. 1990. Grammar engineering: Problems and prospects – Report on the Saarbrücken Grammar Engineering Workshop. Technical Report 1, Saarbrücken, Germany.

Claire Gardent and Laura Kallmeyer. 2003. Semantic construction in FTAG. In *Proceedings of the European chapter of the Association for Computational Linguistics (EACL'03)*, Budapest, Hungary.

Claire Gardent. 2006. Intégration d'une dimension sémantique dans les grammaires d'arbres adjoints. In *Actes de La 13ème édition de la conférence sur le TALN (TALN 2006)*, Leuven, Belgique.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, New York.

Laura Kallmeyer 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*, Tarragona, Spain.

Sabine Lehmann, Stephan Oepen, Sylvie Regnier-Prost, Klaus Netter, Veronika Lux, Judith Klein, Kirsten Falkedal, Frederik Fouvry, Dominique Estival, Eva Dauphin, Hervé Compagnion, Judith Baur, Lorna Balkan, and Doug Arnold. 1996. TSNLP — Test Suites for Natural Language Processing. In *16th International Conference on Computational Linguistics, Proceedings of the Conference*, volume 2, Copenhagen, Denmark.

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

Yannick Parmentier, Laura Kallmeyer, Timm Lichte, and Wolfgang Maier. 2007. XMG: eXtending MetaGrammars to MCTAG. In *Actes de l'atelier sur les formalismes syntaxiques de haut niveau, Conference sur le Traitement Automatique des Langues Naturelles (TALN'2007)*, Toulouse, France.

James Rogers and Vijay Shanker. 1992. Reasoning with descriptions of trees. In *Proceedings of the 30th annual meeting of the Association for Computational Linguistics (ACL 92)*, pages 72–80, Newark, DE.

Sabine Schulte im Walde. 2002. A Subcategorisation Lexicon for German Verbs induced from a Lexicalised PCFG. In *Proceedings of the 3rd Conference on Language Resources and Evaluation*, volume IV, pages 1351–1357, Las Palmas de Gran Canaria, Spain.

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

Anders Søgaard. 2007. *Complexity, expressivity and logic of linguistic theories*. Ph.D. thesis, University of Copenhagen, Copenhagen, Denmark.

XTAG Research Group. 2001. A Lexicalized Tree Adjoining Grammar for English. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA.