XMG: eXtending MetaGrammars to MCTAG

Yannick Parmentier, Laura Kallmeyer, Timm Lichte, Wolfgang Maier

XMG : eXten*ding* MetaGrammars to MCTAG*

Yannick PARMENTIER[1] Laura KALLMEYER[2] Timm LICHTE[2]
Wolfgang MAIER[2]
(1) LORIA - Nancy Université,
Campus Scientifique
BP 239, F-54 506 Vandœuvre-Lès-Nancy Cedex, France
(2) SFB 441 - University of Tübingen,
Nauklerstr. 35, D-72074 Tübingen, Germany
parmenti@loria.fr, lk@sfs.uni-tuebingen.de,
timm.lichte@uni-tuebingen.de, wo.maier@uni-tuebingen.de

Résumé. Dans cet article, nous présentons une extension du système XMG (*eXtensible MetaGrammar*) afin de permettre la description de grammaires d'arbres adjoints à composantes multiples. Nous présentons en particulier le formalisme XMG et son implantation et montrons comment celle-ci permet relativement aisément d'étendre le système à différents formalismes grammaticaux cibles, ouvrant ainsi la voie au multi-formalisme.

Abstract. In this paper, we introduce an extension of the XMG system (eXtensible MetaGrammar) in order to allow for the description of Multi-Component Tree Adjoining Grammars. In particular, we introduce the XMG formalism and its implementation, and show how the latter makes it possible to extend the system relatively easily to different target formalisms, thus opening the way towards multi-formalism.

Mots-clés : Formalismes syntaxiques, grammaires d'arbres, métagrammaires.

Keywords: Syntactic formalisms, tree-based grammars, metagrammars.

1 Introduction

For many NLP applications (*e.g.* generation, machine translation, etc.), large linguistic resources are needed. These resources include (but are not limited to) lexicons and grammars. The latter were originally written by hand. This task of grammar writing was requiring many human resources. Plus, the coherence between the grammatical structures was hard to guarantee (and maintain), as the number of structures / people involved were raising (Erbach & Uszkoreit, 1990). To deal with these issues, several proposals have been made to automatise grammar production. The main proposals are grammar extraction (Xia *et al.*, 2000), grammar inference (Higuera, 2001) and grammar generation. In this paper, we focus on the latter. Grammar generation is based on a formal description of the grammar which is processed to produce a real-size grammar. This technique allows the grammar designer to express linguistic generalisations and

* This work was carried out during a visit of Yannick Parmentier at the SFB 441, University of Tübingen in January 2007.

to test different representation theories. Grammar generation systems can be divided in two main categories, systems based on transformation rules (*e.g. meta-rules* or *lexical rules*) and system based on composition rules (*e.g. metagrammars*).

Transformation rules have been used for many syntactic formalisms such as *Generalized Phrase Structure Grammars* (GPSG), where they are called *meta-rules*. The goal of meta-rules is to build new grammatical structures from existing ones. In unification-based grammars, lexicons are usually defined by associating lexical items with a complex category (represented by a feature-structure). Unlike meta-rules which are applied to grammatical structures, the transformation rules are here applied to lexical entries in order to derive new entries, and are thus called *lexical rules*.

Concerning tree-based grammars, such as *Tree Adjoining Grammars* (TAG), a system of transformation rules has been proposed by (Becker, 1993). In this system, transformation rules are called *meta-rules* and applied to lexical entries containing no longer feature-structures but tree structures (whose nodes may be labelled with feature-structures). In that case, the meta-rules are used to derive new trees. A meta-rule has the following shape : $LHS \rightarrow RHS$, where LHS (respectively RHS) represents the left-hand side (resp. right-hand side) of the rule and consists of a tree fragment. If the left-hand side of the rule matches a given lexical entry, then a tree transformation occurs, replacing the left-hand side in the tree associated with the entry by the right-hand side. For instance, the rule given in Fig. 1 derives the tree fragment for a clitic object starting from the tree fragment for the canonical nominal object.

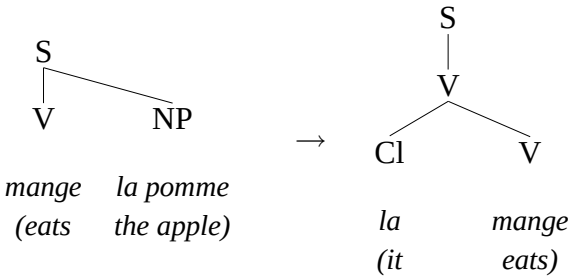

FIG. 1 – Transformation rule for Clitic-Object in French with TAG.

One drawback of such a system comes from the fact that the rule applications must be controlled to avoid over-generation and infinite loops. (Prolo, 2002) proposes to use a declaration (for each lexical entry) of valid application orderings as a control process. Nevertheless, when dealing with real-size grammars, this task of rule ordering may be tedious.

The second trend in grammar generation, *i.e.* systems based on composition rules, has emerged from works on tree-based grammars, especially TAG (Candito, 1996). Here, the factorisation needed to describe a real-size grammar is not provided by transformation rules allowing for the *expansion* of canonical trees. Instead, the factorisation arises from the definition of (i) elementary tree fragments, and (ii) composition rules over these fragments. This factorised definition of the grammatical structures is sometimes called a *metagrammar*. Several metagrammatical systems have been developed, especially for TAG[1]. Among these, one may cite *eXtensible MetaGrammar* (XMG), which distinguishes itself from previous approaches by its extensibility and flexible management of variable scopes (Duchier *et al.*, 2004). On top of providing a high-level language allowing for the description of TAG grammars, the XMG language can be extended

[1]See (Duchier *et al.*, 2004) for a comparison of these systems.

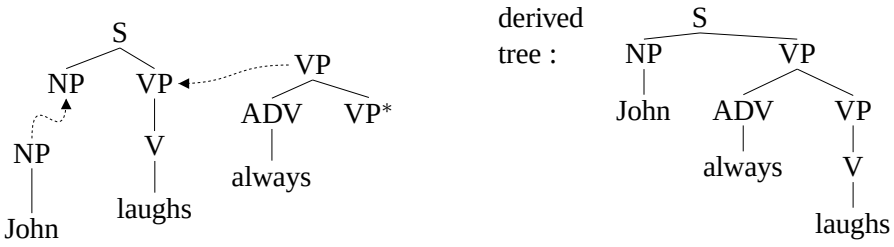

FIG. 2 – TAG derivation for *John always laughs*

to deal with other grammatical formalisms. Such an extension would make it easier to study the common points between *meta*-descriptions for different formalisms, and opens the way towards *multi-formalism*, which is one of the targets of the Mosaïque project[2]. To illustrate this extensibility, we take the example of *Multi-Component Tree Adjoining Grammars* (MCTAG). The paper is organised as follows. In section 2, we introduce MCTAG, motivate their use considering the description of German, and point out the limitations of TAG metagrammatical systems with respect to the description of MCTAG. Then in section 3, we present the XMG formalism and its implementation. In section 4, we show how, concretely, the XMG system has been extended to support the description of MCTAG. Finally, in section 5, we conclude and point out some perspectives for future work.

2 Multi-Component Tree Adjoining Grammars (MCTAG) and "Meta" MCTAG

Tree Adjoining Grammars (TAG) as originally defined by (Joshi *et al.*, 1975) consist of elementary trees which can be combined via *substitution* (replacing a leaf with a new tree) and *adjunction* (replacing an internal node with a new tree). In case of an adjunction, the tree being adjoined has exactly one leaf that is marked as the foot node (marked with an asterisk). Such a tree is called an *auxiliary* tree. When adjoining it to a node n, in the resulting tree, the sub-tree with root n from the old tree is attached to the foot node of the auxiliary tree. Non-auxiliary elementary trees are called *initial* trees. A derivation starts with an initial tree. In a final derived tree, all leaves must have terminal labels. For a sample derivation see Fig. 2.

An extension of TAG that has been shown to be useful for several linguistic applications is Multi-Component TAG (MCTAG) (Joshi, 1987; Weir, 1988). An MCTAG additionally lets one declare tree sets consisting of elementary trees, meaning two things : firstly, using a tree set implicates using all the trees belonging to it ; secondly, the attachment (i.e. adjunction or substitution) of the trees of a tree set can be restricted with respect to the place of attachment : if the trees of a tree set are attached to the same elementary tree, the MCTAG is called *tree-local* ; if they are attached to the same tree set, the MCTAG is called *set-local* ; otherwise (i.e. without attachment restriction) the MCTAG is called *non-local*. Tree-local and set-local MCTAG are polynomially parsable (the former are even strongly equivalent to simple TAG) while non-local MCTAG are NP-complete (Rambow & Satta, 1992).

From a linguistic point of view, MCTAG are particularly interesting when modelling long dis-

[2] *Cf.* `http://mosaique.labri.fr`

tance dependencies and movement effects, since the notion of tree sets allows for a conjoint representation of fillers and gaps in one lexical entry. An early application of tree-local MCTAG was thus dedicated to the modelling of extraposed relative clauses (Kroch & Joshi, 1987). Another field of application is the modelling of scrambling data in German, which furthermore necessitate a TAG formalism more expressive than simple TAG or tree-local MCTAG. Therefore (Rambow, 1994) has provided an MCTAG variant, called V-TAG, that is in fact non-local. However, the desired computational properties are re-implemented via the use of dominance constraints and the admission of non-synchronous attachments of the trees of a tree set. More recently, (Kallmeyer, 2005) has developed another MCTAG variant in order to account for the scrambling data, namely MCTAG with shared nodes (SN-MCTAG). Other than V-TAG, the notion of SN-MCTAG is built upon tree-local MCTAG, but the notion of locality is relaxed, such that non-local attachments are permitted under certain circumstances. Kallmeyer shows that SN-MCTAG is still tractable in polynomial time. As an example a tree-local MCTAG analysis of scrambled constituents in the German Mittelfeld is provided in Fig. 3. Note, that the analysis is the same when using SN-MCTAG.

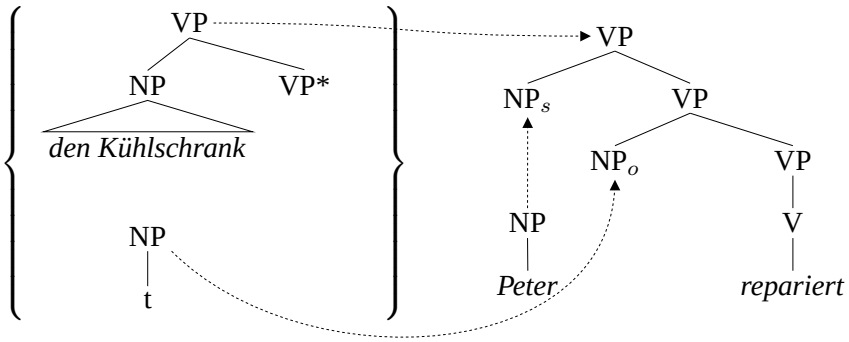

FIG. 3 – A tree-local MCTAG analysis of the German sentence "*[dass] den Kühlschrank Peter repariert*" ('[that] Peter repairs the fridge'). The object noun *den Kühlschrank* ('the fridge') is fronted, leaving a trace in its base position behind the subject NP.

The different MCTAG variants proposed in the literature can be distinguished with respect to the derivations they licence. But independent from this, they all consist of sets of elementary TAG trees. The original metagrammar system XMG did not support tree sets, which made it difficult to use for encoding MCTAG. Indeed, when designing metagrammars for TAG, the user defines tree descriptions whose models are TAG trees. The only grouping of these trees is made through the way the tree descriptions are gathered[3]. In practice, the descriptions are gathered with respect to *sub-categorisation frames* (Crabbé, 2005). In the case of MCTAG, one may want to describe sets of trees according to specific criteria (defined by the metagrammar designer). The XMG language has thus been extended so that the metagrammar designer can define tree descriptions whose models are sets of trees.

3 XMG : an extensible metagrammatical framework

In this section, we introduce the XMG system, and present its main features making it extensible. This introduction will be followed (next section) by a step-by-step presentation of how to extend it to different grammatical formalisms, taking the example of MCTAG.

[3]This gathering is specified using conjunctions and disjunctions, *cf* section 3.

By XMG, we refer to both (i) a metagrammatical formalism, *i.e.* a formal language allowing to express abstractions over the structures of a grammar, and (ii) an implementation of this formalism, in other words, a compiler for the XMG language.

3.1 The XMG formalism

Definition of elementary tree fragments The XMG language allows to describe reusable tree fragments through abstractions called *classes*. A class corresponds to the association of a name with a content :

$$\textit{Class} \quad ::= \quad \textit{Name} \rightarrow \textit{Content} \tag{1}$$

For tree-based grammars, this content corresponds to a tree fragment (*Content*::=*Description*) represented using a tree description logic formula built on the following language :

$$\begin{aligned} \textit{Description} ::= {} & x \rightarrow y \mid x \rightarrow^{+} y \mid x \rightarrow^{*} y \mid x \prec y \mid x \prec^{+} y \mid x \prec^{*} y \mid \\ & x[f{:}E] \mid x(p{:}E) \mid \textit{Description} \wedge \textit{Description} \end{aligned} \tag{2}$$

where x, y are node variables, $\rightarrow$ represents the dominance relation (*mother-of* relation), $\rightarrow^{+}$ its transitive closure, and $\rightarrow^{*}$ its reflexive and transitive closure. $\prec$ refers to the precedence between nodes (*sister-of* relation), $\prec^{+}$ its transitive closure, and $\prec^{*}$ its reflexive and transitive closure. $x[f{:}E]$ is the association of the feature f and the value E to the node referred to by the x variable. $x(p{:}E)$ is the association of the property p and the value E to the x node. Note E is an expression and can correspond to either a variable, a constant or a disjunction over constants (so-called *atomic disjunction*).

Definition of combination of tree fragments Once elementary tree fragments have been defined, it is possible to define combinations over these, using two operators, namely *conjunction* and *disjunction*. Concretely, the XMG language is extended with the following definition :

$$\textit{Content} \quad ::= \quad \textit{Description} \mid \textit{Name} \mid \textit{Content} \vee \textit{Content} \mid \textit{Content} \wedge \textit{Content} \tag{3}$$

The content of a class can either be a description (tree description logic formula) or a name (class instantiation), or a disjunction / conjunction of contents.

This part of the XMG language allows for a flexible control over the class combinations, making it possible to express linguistic properties of natural languages. For instance, the fact that transitive verbs are made of a subject, a verbal morphology (active or passive) and a object can be described within XMG as illustrated below :

$$\textit{transitive} \quad \rightarrow \quad \textit{subject} \wedge \textit{morphology} \wedge \textit{object}$$

To illustrate the expressive power allowed by XMG[4], you can imagine that a subject is not a single tree fragment but a disjunction of tree fragments, each one defining a syntactic realisation of the subject.

An important remark has to be made here. In the above example, nothing is said about the way the tree fragments are "stuck" together, *i.e.* about how nodes are identified. This node identification was a central point in previous metagrammar approaches. In the XMG approach, the scope of a node variable is *by default* local to the class (*i.e.* to the fragment). This means that

[4]Another illustration of this power is the case of the agentless passive. Unlike (Candito, 1996), it does not need any description removal with this language, the description is thus fully declarative.

you can reuse the same node variable in different fragments without any name conflict. When you want to declare that two node variables introduced in different fragments denote the same node, you can use a prefix notation and a node equation :

$$S = subject \ \wedge \ A = morphology \ \wedge \ S.X = A.X$$

Extension to different levels of description Up to now, we have seen how to factorise syntactic information (tree structures) within a metagrammatical description in the XMG language. In order to allow for the extension of different levels of description (such as semantics, or non-tree-based syntax, *etc.*), we have to extend the XMG language. This extension corresponds to the concept of *dimensions*. The content of a class is a description belonging to a given dimension, each dimension has its own sub-language. Definition (3) is extended by :

$$\begin{array}{lll} \textit{Content} & ::= & \textit{Dimension} += \textit{Description} \mid \textit{Name} \mid \\ & & \textit{Content} \vee \textit{Content} \mid \textit{Content} \wedge \textit{Content} \end{array} \quad (4)$$

For instance, XMG integrates a semantic dimension allowing for the description of predicative formulae.

3.2 The XMG compiler

The language introduced above is processed by a compiler in order to produce the grammar described. Before presenting the architecture of this compiler, we can recall that the XMG language is made of two devices :

– a collection of description languages (one for each dimension), allowing for the description of basic units,
– a combination language.

This duality of the language is reflected within the architecture of the compiler, which performs two main tasks : (a) accumulating basic units (in other words, processing the combination rules), and (b) applying a processing on the accumulated units (for instance, once partial tree descriptions are accumulated, computing the corresponding tree models). While the first task is common to all dimensions, the second task is dimension-dependent.

Processing of the combination rules It is worth noticing that the XMG combination language corresponds to a *Definite Clause Grammar* (DCG) (Pereira & Warren, 1980). Indeed, when considering tree descriptions as words, conjunctive and disjunctive rules are just DCG rules. This is why the combination language is processed the same way as a DCG would be by a PROLOG compiler. In the DCG paradigm, one has to define axioms, from which PROLOG computes the corresponding DCG parses. In our case, the metagrammar designer also defines axioms (*i.e.* the classes that encode combination rules leading to total tree descriptions, such as $transitive$ in the above example) using the $value$ keyword. In order to have full control on unification, we decided to develop our own WAM-based virtual machine (see (Duchier *et al.*, 2004)). This virtual machine distinguishes between the different dimensions, thus as an output, it produces a list of accumulated descriptions (total tree descriptions for syntax, list of predicates for semantics).

Additional processing of the accumulated descriptions After the processing of the combination rules, we have a list of descriptions (one description per dimension for each axiom). Let us call this list $L(x) = (D_1, \ldots, D_n)$, where x is an axiom (class name) and D_i the description of the dimension i. Each dimension i is processed by a specific solver S_i, whose role is to produce the models $S_i(D_i)$ of the description D_i.

For the syntactic dimension, say dimension 1, D_1 is a tree description, and the solver S_1 computes all minimal tree models satisfying D_1. Let us briefly introduce S_1. S_1 has been developed as a *Constraint Satisfaction Problem*. The idea behind this is to associate each node variable x of the description D_1 with an integer j, then to define the position of this node in a model as a 5-tuple $N_j^x = (Eq, Up, Down, Left, Right)$ where Eq refers to the node variables (integers) that are identified with x in a model, Up to the node variables that denote the ancestors of x in a model, $Down$ its descendants, etc. Finally the relations between node variables in D_1 are translated into constraints over these 5-tuples, *i.e.* constraints over sets of integers (see (Duchier *et al.*, 2004; Le Roux *et al.*, 2006) for more details).

For the semantic dimension, say 2, D_2 is a list of predicates. There is no need for further processing of this dimension, so S_2 is just the identity operation.

4 Towards a library of operational constraints for describing different target formalisms

Before the work described here took place, the XMG system was supporting the description of TAG, Interaction Grammars (IG)[5], and Hole Semantics. We extended it so that one may also describe MCTAG. This extension was made possible by the modular architecture of the system as advocated in (Le Roux *et al.*, 2006). This extension was made in two steps :

1. extension of the XMG language (either by defining a new dimension with its own sub-language, or by extending an existing one),
2. definition of the solver for this new / extended dimension.

Extension of the XMG language As presented above, MCTAG is an extension of TAG in which the elementary structures of the grammar are sets of trees. In a metagrammatical context, the factorisation of an MCTAG corresponds to the definition of tree fragments that are combined to produce (no more trees but) sets of trees. Concretely, this means that we can keep the same tree description language as the one for TAG given in definition (2). Thus, we do not need a new dimension, we can extend the existing syntactic dimension by adding a unary operator to distinguish between descriptions whose models are trees and those whose models are sets of trees. As introduced in the preceding section, the metagrammar designer defines *axioms* (class names) indicating the classes which refer to total descriptions. These axioms are the starting point for the processing of the combination rules. In our extension, we define a second type of axioms using the $setvalue$ keyword. The classes referred to by these new axioms have to be interpreted as descriptions of sets of trees. This means that we have to define a new solver for these descriptions of sets. Thus, we defined a S_3 solver taking as an input a description belonging to dimension 1 (syntax). While $S_1(D_1)$ computes trees, $S_3(D_1)$ computes sets of trees. Note that S_1 and S_3 can be used within the same metagrammar (*i.e.*, share the same tree fragments).

Definition of a solver for MCTAG While the S_1 solver introduced above applies tree-specific constraints on models (such as the uniqueness of the root node), the S_3 solver we defined for MCTAG behaves differently. S_3 has two major differences compared with S_1[6]. First, there is no

[5] Both TAG and IG were using the same syntactic dimension, as these formalisms are both based on trees.

[6] For lack of space, we do not present neither S_3 nor S_1 in detail here (see (Duchier *et al.*, 2004; Le Roux *et al.*, 2006) for a detailed introduction to S_1).

constraint of root uniqueness, that is to say, two nodes of a model can be such that there is no node above them :

$$\exists\, j, k \in [1..n] \mid N_j^x.Up = \emptyset \wedge N_k^y.Up = \emptyset \wedge (N_j^x.Eq \cap N_k^y.Eq) = \emptyset$$

(n represents the number of node variables in the description). Secondly, two different nodes of a model can belong to two different trees. For our 5-tuple representation, this means that possibly none of their position features (Eq, Up, $Down$, $Left$, $Right$) intersects :

$$\exists\, j, k \in [1..n] \mid (N_j^x.Eq \cup N_j^x.Up \cup N_j^x.Down \cup N_j^x.Left \cup N_j^x.Right) \cap (N_k^y.Eq \cup N_k^y.Up \cup N_k^y.Down \cup N_k^y.Left \cup N_k^y.Right) = \emptyset$$

To illustrate the difference between S_1 and S_3, consider the description A of Fig. 4. This description can be interpreted either as trees ($S_1(A)$), or as sets of trees ($S_3(A)$).

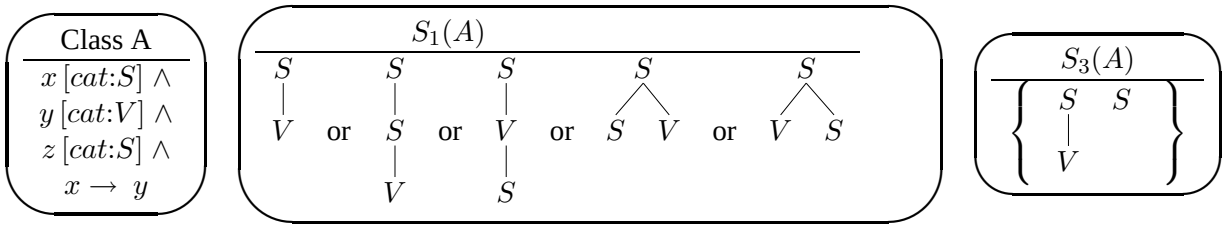

FIG. 4 – Description solving as trees / sets of trees.

When computing trees, S_1 searches for all *minimal* tree models to the description. That is, S_1 does not add any node on top of those referred to by a node variable in the description. S_1 successively tries to identify nodes (when the feature structures labelling the nodes unify), or to add a dominance relation between a node and a local root. S_3 searches for models of sets of trees, more precisely, S_3 does not add any dominance relation on local roots. We have integrated the S_3 solver in the current XMG compiler[7], and we started implementing a metagrammar for German using this new multi-component dimension.

Note that when the metagrammar designer specifies a precedence (or dominance) relation between nodes belonging to distinct trees of a set, S_3 is unable to compute a solution. Such relations between nodes of elements of a set are used in some extensions of MCTAG, and corresponds to (unsolved) constraints on the trees of the set. These constraints have to be applied during parsing. It would be interesting to extend the XMG language to include node relations that are not to be solved.

Towards multi-formalism We have seen a first extension of the XMG system to deal with MCTAG. As this formalism is based on trees, we did not need a new dimension. Nevertheless, it is worth noticing that such an extension was facilitated by the modular architecture of the system (virtual machine processing DCG rules and solver computing grammatical structures). To sum up, it is possible to extend XMG for compiling a specific dimension provided you define a language for describing it, and a solver for interpreting the corresponding description. The latter can be seen as a set of operational constraints applied to a description in order to produce valid structures (with respect to grammatical criteria).

[7] In the XMG-Tuebingen development branch of the subversion repository, see `http://sourcesup.cru.fr/xmg`.

The next step in this work is to define a library of solvers applying specific operational constraints on grammatical descriptions. These constraints will be selected dynamically by the metagrammar designer depending on the targeted grammatical formalism. Such a library would allow the metagrammar designer to abstract away from the technical aspects of a given grammatical formalism, providing him with a high-level description language. Furthermore, the metagrammar designer would be able to define a linguistic description that would be interpreted by different solvers to produce grammars for different formalisms (*i.e.*, multi-formalism).

A second interesting perspective consists of the development of a device allowing for solver specification. Thus the linguist would be able to define its own grammatical criteria from which the corresponding solver would be generated automatically.

Finally, another interesting perspective concerns parsing directly from the metagrammatical descriptions (*i.e.*, without computing the elementary units of the grammar). This path is followed by the MGCOMP system (Villemonte de la Clergerie, 2005).

5 Conclusion and Perspectives

This paper addresses a central problem of large coverage grammar implementation, namely the difficulty to keep the grammar consistent across its different parts in spite of the considerable redundancy that arises with the increasing size of an electronic grammar. This problem is particularly prominent in lexicalised grammars that do not allow to formulate linguistic generalisations outside the lexical entries. As a solution, eXtensible MetaGrammar (XMG) provides a platform for grammar development that allows to factorise the lexical entries of a grammar into smaller pieces that can then be used in different places. XMG is intended for lexicalised tree grammars, in particular Tree Adjoining Grammars (TAG). TAG allows to describe a large range of linguistic phenomena, in some cases however its expressive power is too limited. One such example is the phenomenon of scrambling in so-called free word order languages such as German. The different proposals for extending TAG in order to account for German scrambling data all have in common that they use an MCTAG, i.e., a grammar consisting of sets of trees instead of trees. The goal of this paper was to extend XMG so that it can be used not only to describe TAG but also MCTAG.

In the paper, we have achieved such an extension by (optionally) relaxing the conditions on the models, in particular omitting the assumption about the uniqueness of the root node. In that case, a minimal model for a given description in the metagrammar might still be a tree (if all nodes are connected in the description) but it could also be a set of disconnected trees. We think that our technique of relaxing the restriction of XMG to tree models opens up interesting perspectives for future work oriented towards other grammar formalisms. The idea to allow graphs different from proper trees as models could be exploited for example for formalisms involving feature structures instead of trees.

Such an extension of metagrammars to different target formalisms would allow to study how the factorisation and the expression of linguistic generalisation is represented in these formalisms (this comparative task would be facilitated by using the same language and system). The results of such a study would make it possible to go further towards *strong* multi-formalism, that is to say towards the compilation of different grammars (*i.e.* in different formalisms) starting from a single meta-description.

Références

BECKER T. (1993). *HyTAG : A new Type of Tree Adjoining Grammars for Hybrid Syntactic Representation of Free Word Order Language*. PhD thesis, Universität des Saarlandes.

CANDITO M. (1996). A principle-based hierarchical representation of LTAGs. In *Proceedings of COLING'96, Kopenhagen*.

CRABBÉ B. (2005). *Représentation informatique de grammaires fortement lexicalisées : Application à la grammaire d'arbres adjoints*. PhD thesis, Université Nancy 2.

DUCHIER D., LE ROUX J. & PARMENTIER Y. (2004). The Metagrammar Compiler : An NLP Application with a Multi-paradigm Architecture. In *2nd Internationale Conference of Mozart/Oz users (MOZ'2004)*, Charleroi.

ERBACH G. & USZKOREIT H. (1990). *Grammar Engineering : Problems and Prospects – Report on the Saarbrücken Grammar Engineering Workshop*. Rapport interne 1, Saarbrücken, Germany.

HIGUERA C. D. L. (2001). Current trends in grammatical inference. *Lecture Notes in Computer Science*, **1876**.

JOSHI A. K. (1987). An introduction to tree adjoining grammars. In A. MANASTER-RAMER, Ed., *Mathematics of Language*, p. 87–114. John Benjamins, Amsterdam.

JOSHI A. K., LEVY L. S. & TAKAHASHI M. (1975). Tree adjunct grammars. *Journal of Computer and System Science*, **10**, 136–163.

KALLMEYER L. (2005). Tree-local multicomponent tree adjoining grammars with shared nodes. *Computational Linguistics*, **31 :2**, 187–225.

KROCH A. S. & JOSHI A. K. (1987). Analyzing extraposition in a tree adjoining grammar. In G. J. HUCK & A. E. OJEDA, Eds., *Discontinous Constituency*, number 20 in Syntax and Semantics, p. 107–149. Academic Press, Inc.

LE ROUX J., CRABBÉ B. & PARMENTIER Y. (2006). A constraint driven metagrammar. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+8), Sydney, Australia*.

PEREIRA F. & WARREN D. (1980). Definite clause grammars for language analysis —a survey of the formalism and a comparison to augmented transition networks. *Artificial Intelligence*, **13**, 231–278.

PROLO C. A. (2002). Generating the XTAG English grammar using metarules. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'2002)*, p. 814–820, Taipei, Taiwan.

RAMBOW O. (1994). *Formal and Computational Aspects of Natural Language Syntax*. PhD thesis, University of Pennsylvania, Philadelphia. IRCS Report 94-08.

RAMBOW O. & SATTA G. (1992). Formal properties of non-locality. In *Proceedings of 1st International Workshop on Tree Adjoining Grammars*.

VILLEMONTE DE LA CLERGERIE E. (2005). DyALog : a tabular logic programming based environment for NLP. In *Proceedings of CSLP'05*, Barcelona.

WEIR D. J. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania.

XIA F., PALMER M. & JOSHI A. (2000). A Uniform Method for Grammar Extraction and Its Application. In *Proceedings of 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.