XMG: a Multi-formalism Metagrammatical Framework

Yannick Parmentier, Joseph Le Roux

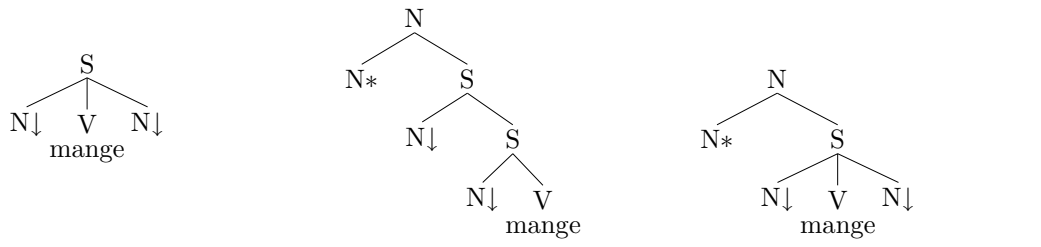
XMG: a Multi-formalism Metagrammatical Framework

Yannick Parmentier - Joseph Le Roux
Langue Et Dialogue / Calligramme Projects - INRIA / LORIA
615, Rue du Jardin Botanique
54 600 Villers-Lès-Nancy, France
{parmenti,leroux}@loria.fr

Abstract. In this paper we introduce XMG[1] (*eXtensible MetaGrammar*), a system dedicated to the production of wide coverage lexicalised grammars. In particular, we show that XMG provides a representation language suitable for describing different linguistic dimensions and different grammatical formalisms. Furthermore, we briefly sketch the architecture of the XMG compiler showing that it encodes a theoretically sound processing of the XMG formalism.

1 Introduction

We are concerned with the production of grammars for strongly lexicalised formalisms. In such formalisms, the linguistic knowledge is included in the lexicon, which can be seen as a function mapping words to the set of grammatical structures reflecting their usages in sentences. So, in order to get a realistic coverage of natural languages, this mapping must reflect as many behaviours of the word as possible (*e.g.* interrogative, active, passive...). For instance, in Tree Adjoining Grammars (TAG), the word *mange* (eats) is associated with the following structures (among others):



Jean mange une pomme / *John eats an apple*
La pomme que Jean mange / *The apple that John eats*
Jean qui mange une pomme / *John who eats an apple*
...

[1] freely available at `http://sourcesup.cru.fr/xmg`.

Proceedings of the Tenth ESSLLI Student Session
Judit Gervain (editor)

We note that in such lexicons:

1. a given structure can be associated with several words (*e.g.* a large number of structures are shared by transitive verbs) ;
2. structures have many fragments in common (*e.g.* the S–V chunk on the above structures).

This redundancy leads to the following problems: (a) it is hard to preserve consistency between grammatical structures when changes[2] are made, and (b) we cannot express linguistic generalisations.

In order to avoid such drawbacks, one would like to automatically produce the grammar from a highly factorised description of the linguistic concepts underlying the grammar (the *metagrammar*).

The paper is structured as follows. First (section 2), we show how a wide coverage TAG can be automatically produced with XMG. This includes the presentation of XMG's representation language. We then compare this work with existing approaches. Secondly (section 3), we show how to extend XMG to cover other linguistic dimensions (*e.g.* semantics). Finally we present the modular architecture of XMG.

2 Production of wide coverage TAG with XMG

In this section, we introduce the XMG formalism[3] showing how it can be used to describe the main linguistic and formal properties encoded in a TAG[4] and comparing it with existing approaches to factorising TAGs.

2.1 XMG's core language

A wide coverage TAG is composed of thousands of trees. To avoid redundancy, the first improvement is to consider not complete trees but tree fragments. Each fragment is represented by means of a tree description language (see (Rogers and Vijay-Shanker 1992)). Furthermore, these fragments can reuse others, for instance by means of inheritance as presented in (Vijay-Shanker and Schabes 1992). Then, these fragments are combined to build TAG trees. Two issues have to be solved: (a) how to split the trees into fragments to reach a good factorisation, and (b) how to control the combination of these fragments to produce the appropriate trees.

The XMG formalism tackles these issues by providing a representation language where tree fragments can be (1) referred to by **abstractions**, and (2) combined using **conjunctive** and **disjunctive** composition.

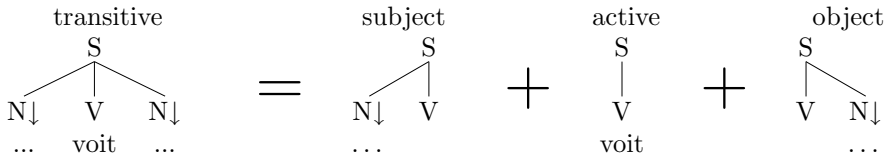
[2] such as the modification of the representation of the verb agreement.

[3] We present the XMG abstract language, for lack of space we do not introduce the high-level concrete syntax, please see (Duchier et al. 2004).

[4] Note that we do not introduce the TAG grammatical formalism here, please see (Joshi and Schabes 1997) for such an introduction.

Representation of the combinations of fragments One important underlying idea in XMG is that the combination of pieces of information in a metagrammatical description can be compared with rewriting rules.

To illustrate this, let us consider the combination of the tree fragments used to produce the tree associated with the French lexical item voit (*sees*), *i.e.* a transitive verb. This combination involves 3 fragments, namely the one representing the subject, the one representing the verbal morphology (active) and the one for the object:



Another way to state this is that the combination of *subject*, *active*, and *object* can be rewritten as *transitive*:

$$transitive \quad \rightarrow \quad subject \wedge active \wedge object$$

This rewriting system corresponds to the formalism of *Definite Clause Grammars* (DCG) with a difference: terminal symbols are not limited to words, but can be tree fragments. A DCG performs an *accumulation* of the terminal symbols encountered during the derivation (*i.e.* application of the rewriting rules).

The language used to describe combinations of fragments in XMG is defined by:

$$Clause \quad ::= \quad Name \rightarrow Goal \tag{1.1}$$

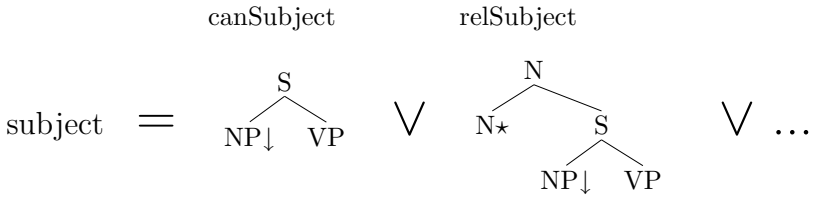
$$Goal \quad ::= \quad Description \mid Name \mid Goal \vee Goal \mid Goal \wedge Goal \tag{1.2}$$

$$Query \quad ::= \quad Name \tag{1.3}$$

As mentioned above, this language includes **abstraction** (1.1) that allows one to reuse a fragment, **conjunction** and **disjunction** (1.2). With such a language, one can define precisely and flexibly the combinations necessary to build the grammar. For instance, one can refine the *transitive* example by stating that the subject can be *realised* in different ways, such as a *canonical subject*, a *relativised subject*, etc:

$$subject \quad \rightarrow \quad canSubject \vee relSubject \vee \ldots$$

That is, the following tree fragments describe the possible realisations of a subject:



Note that this can be done thanks to the *indeterminism* provided by the disjunction operation.

During compilation, the XMG system computes all derivations of the corresponding DCG, starting from a *query* (1.3). With the above example, the query would be *transitive* and the associated trees would be described by the following clause, where each conjunction corresponds to a solution to the query:

$$\begin{array}{lll} \mathit{transitive} & \rightarrow & (\mathit{canSubject} \wedge \mathit{active} \wedge \mathit{object}) \\ & \vee & (\mathit{relSubject} \wedge \mathit{active} \wedge \mathit{object}) \\ & \vee & \ldots \end{array}$$

Thus, we obtain for each query the enumeration of all satisfying tree descriptions.

Representation of the content of the fragments With XMG, the tree fragments of a TAG metagrammar are expressed by means of a tree description language including the following operators:

$$\begin{array}{rcl} \mathit{Description} & ::= & x \rightarrow y \mid x \rightarrow^{+} y \mid x \rightarrow^{*} y \mid x \prec y \mid x \prec^{+} y \mid \\ & & x \prec^{*} y \mid x[f{:}E] \mid x(p{:}E) \end{array} \tag{1.4}$$

where x, y represent node variables, $\rightarrow$ immediate dominance, $\rightarrow^{+}$ strict dominance, $\rightarrow^{*}$ large dominance, $\prec$ is immediate precedence, $\prec^{+}$ strict precedence, and $\prec^{*}$ large precedence. $x[f{:}E]$ constrains feature f with associated expression E on node x (a feature can for instance refer to the *syntactic category* of the node), while $x(p{:}E)$ specifies its property p (node properties are used to add control on fragment combinations, see section 4). An expression E can either be a constant, or a variable, or a complex structure.

Note that these descriptions contain variables that can refer not only to nodes, but also to feature values or node properties, and that these variables can be shared with other fragments.

Information sharing between fragments As mentioned above, the descriptions of tree fragments can use variables. In other words, the descriptions introduced in (1.2) and therefore the clauses in (1.1) contain variables. By default, the scope of these variables is limited to the clause. Nevertheless

the XMG language supports the management of variable scope by means of an *export* concept. An export defines which variables are visible when the clause is called. More precisely, an export associates a dedicated feature structure containing the exported variables with the clause. Thus (1.1) is extended in the following way:

$$Clause \quad ::= \quad \langle V_1, \ldots, V_n \rangle \Leftarrow Name \quad \rightarrow \quad Goal \tag{1.5}$$

and conversely, the *Goal* expression (1.2) is extended to specify that at each invocation of a clause, the exported feature structure is made accessible via an identifier (here *Var*):

$$\begin{array}{lll} Goal & ::= & Description \mid Var \Leftarrow Name \mid \\ & & Goal \vee Goal \mid Goal \wedge Goal \end{array} \tag{1.6}$$

Reusing a variable V_i of the fragment whose abstraction is named *Name* can then be done by using the *dot* operation: $Var.V_i$.

For example, let us consider the class *classA* containing 2 nodes x and y with x dominating y, both nodes being *exported*.

$$\langle x, y \rangle \Leftarrow classA \quad \rightarrow \quad (x \rightarrow y\ )$$

A second class *classB* may access any of them as illustrated below, where the category of the x node is specified inside the *classB* fragment:

$$classB \quad \rightarrow \quad (A \Leftarrow classA \quad \wedge \quad A.x[cat : s])$$

2.2 Comparison with related work

In the field of compact representation for lexicalised TAGs, two trends have emerged:

1. systems based on **lexical rules**,
2. systems based on **fragments and combinations** (*i.e.* metagrammars).

Thus, (Becker 2000) uses lexical rules (called *metarules*) to produce automatically the trees of a TAG. One drawback of this approach is that it leads to the definition of complex ordered application schemes (see (Prolo 2002))[5].

The second trend was first investigated by (Candito 1996). She used the ideas of fragments and combinations along with a structuring of the fragment inheritance hierarchy according to linguistic motivations. The combining process follows an algorithm dependent on this structuring. One important

[5]Nevertheless a broad-coverage HPSG, which is based on lexical rules, exists for English, along with a development kit containing generalisations over the components of such a grammar. We do not detail these results here, please see (Copestake and Flickinger 2000) and (Bender et al. 2002).

drawback of this proposal is that her representation language makes use of global names to refer to nodes, so that the same tree fragment cannot be reused twice within the same call.

Close to Candito's approach, (Xia et al. 1998) propose an abstract representation of the lexicon using fragments (called *blocks*). Xia's proposal is more flexible than Candito's as the blocks can be arbitrarily combined. Nevertheless her representation language suffers from a lack of expressivity and thus overgenerates (see (Crabbé and Duchier 2004)).

Note that all these approaches are closely linked to the TAG formalism. None of them have been used in a multi-formalism context.

The first attempt to provide a flexible metagrammatical framework is (Gaiffe et al. 2002). The authors depart from Candito's approach by separating the fragment specification and the combination process. The latter is controlled by means of *needs and resources*. Hence Gaiffe's metagrammar compiler has been used to produce automatically *lexical functional grammars* (LFG, see (Clément and Kinyon 2003)). However, this production of LFGs corresponds to a diverted use of the compiler by decorating TAG trees with functional annotations. After compilation of the metagrammar, TAG *trees* are produced, that need to be interpreted to extract the LFG *rules*. Even if the linguistic properties are encoded at a metagrammatical level, why not make the metagrammar compiler generate the rules ? Another point is that this approach still makes use of global names, so that (1) the development of real size grammars remains difficult (*cf* name conflicts), (2) it does not provide an efficient way to deal with argument deletion (*e.g.* passive without agent), and (3) one cannot reuse the same fragment several times (*e.g.* verbs with 2 prepositional phrases).

Recently, a new metagrammatical framework has been developed by Thomasset and De La Clergerie (see (Thomasset and Villemonte de la Clergerie 2005)). A key point of this approach is that the metagrammar can produce *factorised trees*, thus allowing a better structure sharing (compact grammar). But, we do not have information yet concerning the usability of this system.

3 From TAG to multi-formalism

In order to reach a certain degree of extensibility, we need to be able to describe not only tree descriptions but also other levels of linguistic description, each with its own representation (*e.g.* attribute-value matrices (AVMs), semantic formulas, etc).

To support such a multi-level description, the combination must process *distinguished* types of description which we will call **dimensions**. Above, we compared the metagrammar with a DCG, where tree fragments are *accumulated* instead of words. Following this idea, we can refer to the formalism

of *Extended Definite Clause Grammars* (EDCG, see (Van Roy 1990)) to have several named accumulators. Thus we will be able to manage several dimensions. More precisely, we extend our representation language by replacing *Description* in (1.6) with *Dimension+=Description*:

$$\begin{array}{lll} \mathit{Goal} & ::= & \mathit{Dimension}{+=}\mathit{Description} \mid \mathit{Var} \Leftarrow \mathit{Name} \mid \\ & & \mathit{Goal} \vee \mathit{Goal} \mid \mathit{Goal} \wedge \mathit{Goal} \end{array} \quad (1.7)$$

where $+=$ is an accumulation operation, its semantics depends on the dimension processed (*e.g.* conjunction).

At the time of this writing, 3 dimensions are implemented in the XMG system, namely **syn** for syntactic descriptions (based on tree descriptions, presented above), **sem** for semantic representations (based on *Hole Semantics*, see (Gardent and Kallmeyer 2003)), and **dyn** corresponding to an open AVM whose role is double: (a) allowing to access some information in the **syn** or **sem** dimensions through coindexation, and (b) associating specific information to the grammar entries (such as morpho-syntactic information).

Therefore, we can define classes containing tree fragments and other containing semantic information. Then we can specify a syntax / semantics interface by using the dimension **dyn** to share unification variables between **syn** and **sem**. On the figure 1.1, we represent the syntactic and semantic information for intransitive verbs (*e.g.* tree and unary relation), and then use **dyn** to state that the subject corresponds to the argument of the semantic relation.

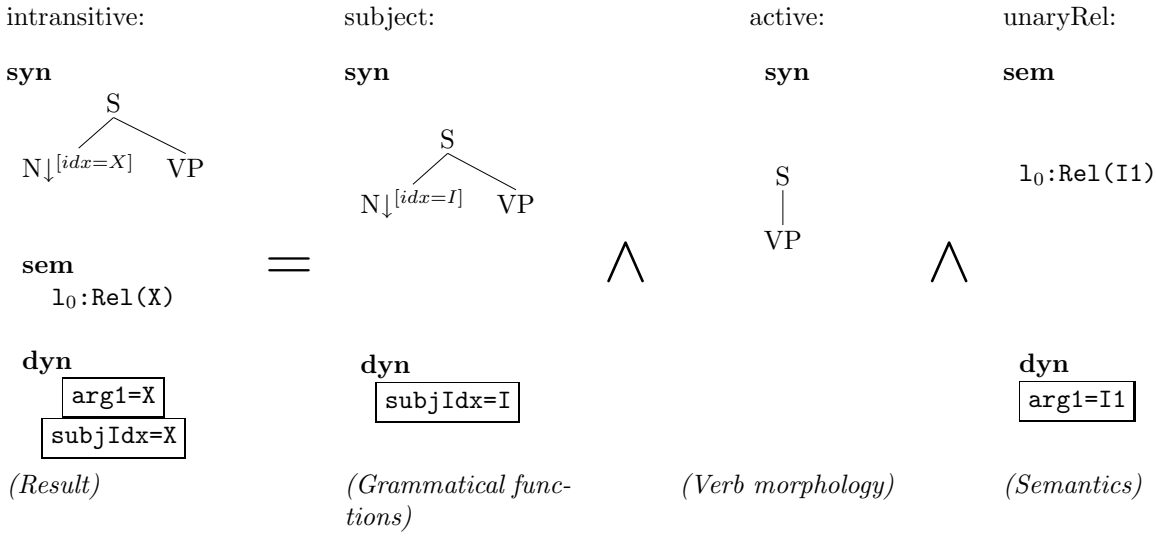


Figure 1.1: A multi-dimensional fragment of the metagrammar

By using the **syn** and **dyn** dimensions, we have been able to automatically produce an Interaction Grammar (IG, (Perrier 2003)) which is currently

used within the LEOPAR parsing system[6]. Similarly, we develop a wide coverage French TAG which encodes both syntactic and semantic information (see (Gardent and Parmentier 2005)).

XMG can furthermore be extended to other syntactic or semantic formalisms by defining new dimensions, each described by a specific language. The XMG user would be able to **dynamically** define a metagrammatical framework that suits its target formalism by loading the adequate dimensions.

4 A modular architecture

In this section, we present the architecture of XMG. Indeed, an important feature of the XMG approach is that metagrammars are processed in the same way as artificial languages. More specifically, the XMG system is composed of 3 parts[7], namely:

- a **compiler** that parses XMG's concrete syntax to produce core language code ;
- a **virtual machine** (VM) performing *accumulation* of dimensions, along with *unification*. This VM is inspired by the *Warren's Abstract Machine* (WAM, see (Ait-Kaci 1991)). As in EDCGs, the VM computes the derivations by evaluating queries ;
- a **third part** for additional processings of the accumulated structures.

This third part is completely modular, that is to say users can chose which modules they need to include. These modules perform various tasks on the accumulated structures:

Resolution of descriptions The VM yields as output a snapshot of its accumulators for each successful derivation, say $(D_1, ..., D_n)$ for n dimensions. For instance, in the D_1 dimension (**syn**), this snapshot corresponds to tree descriptions. In the TAG formalism the elements of the grammar are trees. Hence, the structures produced by the VM (*i.e.* tree descriptions) need to be further processed so that we obtain trees. To complete this processing, we use a description solver such as the one introduced in (Duchier 2000). This solver is implemented through constraints on set of integers. First, each node of the description is assigned an integer. Then, we define for each node N^i the sets N^i_{Eq}, N^i_{Up}, N^i_{Down}, N^i_{Left}, N^i_{Right} representing respectively the nodes that are unified with N^i, above N^i in the model, below N^i, on the left, and on the right of N^i. Finally, the relations between nodes are

[6] freely available at `http://www.loria.fr/equipes/calligramme/leopar/`

[7] A more detailed presentation of XMG's architecture is given in (Duchier et al. 2004).

represented by means of constraints on these sets of integers. For instance, if a node N^i dominates a node N^j, then the following constraint holds[8]:

$$N^i \rightarrow N^j \equiv [N^i_{EqUp} \subseteq N^j_{Up} \wedge N^i_{Down} \supseteq N^j_{EqDown} \wedge N^i_{Left} \subseteq N^j_{Left} \wedge N^i_{Right} \subseteq N^j_{Right}]$$

$$\text{where} \qquad N^i_{EqUp} = N^i_{Eq} \cup N^i_{Up} \quad \textit{and} \quad N^j_{EqDown} = N^j_{Eq} \cup N^j_{Down}$$

That is, the set of nodes that are above a mother node is included in the set of nodes that are *strictly* above its daughter node *and* the set of nodes that are below a mother node contains the set of nodes that are *strictly* below its daughter node *and* the set of nodes that are on the left (respectively the right) of the mother node is included in the set of those that are on the left (respectively right) of the daughter node.

Actually each dimension is processed by a solver computing its realisation according to a given predicate R_i (that may be equality). Note that, since dimensions may share variables, we want all simultaneous solutions of $(R_1(D_1), \ldots, R_n(D_n))$.

Extended control on combinations of fragments When developing a metagrammar for a wide coverage grammar, one may want to constrain the fragment combining semi-automatically, *i.e.* without having to define every node equality. This can be done for instance by using a **resource sensitivity tree language** whose role is to prevent some combinations and force others. For instance such a language can be a colour language, where the elements are (red, black, white), and the combination rules[9]:

	$\bullet_B$	$\bullet_R$	$\circ_W$	$\bot$
$\bullet_B$	$\bot$	$\bot$	$\bullet_B$	$\bot$
$\bullet_R$	$\bot$	$\bot$	$\bot$	$\bot$
$\circ_W$	$\bullet_B$	$\bot$	$\circ_W$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

Thus, by labelling the nodes of the description with adequate colours, we can prevent some node from merging and force others to do so. Technically, the tree descriptions output by the VM are solved by an extended solver, which corresponds to the TAG solver introduced above, coupled with a specific module for colour constraints solving. The details of the use of such a language to prevent tree overgeneration when producing a TAG are given in (Crabbé and Duchier 2004) .

Furthermore XMG makes use of additional operations to constrain the produced structures. For example, these structures may be filtered according to linguistic principles such as the unicity of extraction in French

[8] See also (Duchier and Niehren 2000) for a detailed presentation.

[9] $\bot$ represents failure.

(see (Crabbé and Duchier 2004)). Actually, a library of constraining properties has been implemented. The metagrammar designer may select and *parameterise* the operations needed (at the time of this writing, namely *unicity(X)* or *rank* for clitics ordering).

Formatting of the output As a result of a metagrammar compilation, XMG produces entries of a grammar (at the moment, trees for TAG and tree descriptions for IG). These entries can either be printed through a graphical user interface or translated to an XML format, so that they can be easily used by NLP tools such as natural language generators or parsers.

5 Conclusion

We have presented here a new metagrammatical framework that can supports several grammatical formalisms (TAG and IG at the moment).

The XMG system is freely available at `http://sourcesup.cru.fr/xmg` under the terms of the CeCILL license[10]. It has been developed in Oz/Mozart[11]. The supported platforms are Linux, Mac and Windows.

XMG has been used successfully to develop a wide coverage TAG for French (see (Crabbé 2005a), (Gardent and Parmentier 2005), and (Crabbé 2005b)) and a medium size IG. This TAG metagrammar, containing 285 classes, produces about 5,000 *non-anchored* TAG trees (that is, tree schematas where a node is distinguished to receive the lexical item). It is currently evaluated in syntactic parsing on the TSNLP[12]. The first results are encouraging since the success rate is about 75%. To give an overview of the efficiency of the system, it takes 5 minutes to compile this grammar on a Pentium 4 - 2.66 Ghz processor with 1 Go RAM.

We plan to develop a library of dimensions, each equipped with a specific language. This will allow the description of an arbitrary number of grammatical formalisms by using adequate dimensions.

We are also working on the use of the automatically produced wide coverage TAG with semantic information in the context of parsing (Gardent and Parmentier 2005) and generation (Gardent and Kow 2004).

Acknowledgements

We are grateful to Benoît Crabbé, Denys Duchier, Claire Gardent, Guy Perrier, and Sébastien Hinderer for useful comments on this work.

[10] See `http://www.cecill.info` for more details on this license, which follows the principles of the GNU GPL.

[11] See `http://www.mozart-oz.org`.

[12] See `http://tsnlp.dfki.uni-sb.de/tsnlp/`.

Bibliography

Ait-Kaci, H. (1991). Warren's Abstract Machine: A Tutorial Reconstruction. In K. Furukawa (Ed.), *Logic Programming: Proc. of the Eighth International Conference*. Cambridge, MA: MIT Press.

Becker, T. (2000). Patterns in metarules. In A. Abeille and O. Rambow (Eds.), *Tree Adjoining Grammars: formal, computational and linguistic aspects*. CSLI publications, Stanford.

Bender, E., D. Flickinger, and S. Oepen (2002). The Grammar Matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proceedings of COLING 2002 Workshop on Grammar Engineering and Evaluation, Taipei, Taiwan*.

Candito, M. H. (1996). A principle-based hierarchical representation of LTAGs. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING'96), Kopenhagen*.

Clément, L. and A. Kinyon (2003). Generating LFGs with a MetaGrammar. In *Proceedings of the 8th International Lexical Functional Grammar Conference, Saratoga Springs, NY*.

Copestake, A. and D. Flickinger (2000). An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000), Athens, Greece*.

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

Crabbé, B. (2005b). *Représentation informatique de grammaires fortement lexicalisées - Application à la grammaire d'arbres adjoints*. Ph. D. thesis, Université Nancy 2. To appear.

Crabbé, B. and D. Duchier (2004). Metagrammar Redux. In *International Workshop on Constraint Solving and Language Processing - CSLP 2004, Copenhagen*.

Duchier, D. (2000). Constraint Programming For Natural Language Processing. Lecture Notes, ESSLLI 2000. Available at http://www.ps.uni-sb.de/Papers/abstracts/duchier-esslli2000.html.

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

Duchier, D. and J. Niehren (2000). Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic (CL2000)*, Volume 1861 of *Lecture Notes in Computer Science*, pp. 326–341. Springer.

Gaiffe, B., B. Crabbé, and A. Roussanaly (2002). A New Metagrammar Compiler. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6), Venice*.

Gardent, C. and L. Kallmeyer (2003). Semantic construction in FTAG. In *Proceedings of the 10th meeting of the European Chapter of the Association for Computational Linguistics, Budapest*.

Gardent, C. and E. Kow (2004). Génération et sélection de paraphrases grammaticales. In *journée ATALA sur la génération de Langue Naturelle, Paris*.

Gardent, C. and Y. Parmentier (2005). Large scale semantic construction for Tree Adjoining Grammars. In *Proceedings of the Fifth International Conference on Logical Aspects of Computational Linguistics (LACL05), Bordeaux*.

Joshi, A. and Y. Schabes (1997). Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Volume 3, pp. 69 – 124. Springer, Berlin, New York.

Perrier, G. (2003). Les grammaires d'interaction. Habilitation à diriger les recherches en informatique, Université Nancy 2.

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

Rogers, J. and K. Vijay-Shanker (1992). Reasoning with descriptions of trees. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics, pp. 72 - 80*.

Thomasset, F. and E. Villemonte de la Clergerie (2005, June). Comment obtenir plus des méta-grammaires. In *Proceedings of TALN'05*, Dourdan, France. ATALA. To appear.

Van Roy, P. (1990). Extended DCG Notation: A Tool for Applicative Programming in Prolog. Technical report, Technical Report UCB/CSD 90/583, Computer Science Division, UC Berkeley.

Vijay-Shanker, K. and Y. Schabes (1992). Structure sharing in lexicalized tree adjoining grammars. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'92), Nantes, pp. 205 - 212*.

Xia, F., M. Palmer, K. Vijay-Shanker, and J. Rosenzweig (1998). Consistent Grammar Development Using Partial-Tree Descriptions for Lexicalized Tree Adjoining Grammar. *Proceedings of TAG+4*.