XMG : eXtensible MetaGrammar

Benoît Crabbé[*]
INRIA - Université Paris 7

Denys Duchier[**]
LIFO - Université d'Orléans

Claire Gardent[†]
CNRS - LORIA, Nancy

Joseph Le Roux[‡]
LIPN - Université Paris Nord

Yannick Parmentier[§]
LIFO - Université d'Orléans

In this article, we introduce eXtensible MetaGrammar (XMG), a framework for specifying tree-based grammars such as Feature-Based Lexicalised Tree-Adjoining Grammars (FB-LTAG) and Interaction Grammars (IG). We argue that XMG displays three features which facilitate both grammar writing and a fast prototyping of tree-based grammars. Firstly, XMG is fully declarative. For instance, it permits a declarative treatment of diathesis that markedly departs from the procedural lexical rules often used to specify tree-based grammars. Secondly, the XMG language has a high notational expressivity in that it supports multiple linguistic dimensions, inheritance and a sophisticated treatment of identifiers. Thirdly, XMG is extensible in that its computational architecture facilitates the extension to other linguistic formalisms. We explain how this architecture naturally supports the design of three linguistic formalisms namely, FB-LTAG, IG, and Multi-Component Tree-Adjoining Grammar (MC-TAG). We further show how it permits a straightforward integration of additional mechanisms such as linguistic and formal principles. To further illustrate the declarativity, notational expressivity and extensibility of XMG, we describe the methodology used to specify an FB-LTAG for French augmented with a unification-based compositional semantics. This illustrates both how XMG facilitates the modelling of the tree fragment hierarchies required to specify tree-based grammars and of a syntax/semantics interface between semantic representations and syntactic trees. Finally, we briefly report on several grammars for French, English and German that were implemented using XMG and compare XMG to other existing grammar specification frameworks for tree-based grammars.

[*] UFR de Linguistique, Université Paris 7, Case 7003, 2, Place Jussieu, F-75251 Paris Cedex 05, France. E-mail: bcrabbe@linguist.jussieu.fr.

[**] Laboratoire d'Informatique Fondamentale d'Orléans, Bâtiment IIIA, Rue Léonard de Vinci, B.P. 6759, F-45067 Orléans Cedex 2, France. E-mail: denys.duchier@univ-orleans.fr.

[†] Laboratoire LORIA - CNRS, Projet Talaris, Bâtiment B, BP 239, Campus Scientifique, F-54506 Vandœuvre-Lès-Nancy Cedex, France. E-mail: gardent@loria.fr.

[‡] Laboratoire d'Informatique de Paris Nord, UMR CNRS 7030, Institut Galilée - Université Paris-Nord, 99, avenue Jean-Baptiste Clément, F-93430 Villetaneuse , E-mail: leroux@univ-paris13.fr

[§] Laboratoire d'Informatique Fondamentale d'Orléans, Bâtiment IIIA, Rue Léonard de Vinci, B.P. 6759, F-45067 Orléans Cedex 2, France. E-mail: yannick.parmentier@univ-orleans.fr.

Submission received: 27 March 2009
Revised version received: 2 July 2012
Accepted for publication: 11 August 2012

1. Introduction

In the late eighties and early nineties, many grammar engineering environments were developed to support the specification of large computational grammars for natural language. One may for instance cite XLE (Kaplan and Newman 1997) for specifying Lexical-Functional Grammars (LFG), LKB (Copestake and Flickinger 2000) for specifying Head-driven Phrase Structure Grammars (HPSG), and DOTCCG (Baldridge et al. 2007) for specifying Combinatory Categorial Grammars (CCG). Concretely, such environments usually rely on (i) a formal language used to describe a target computational grammar, and (ii) a processor for this language, which aims at generating the actual described grammar (and potentially at checking it, *e.g.*, by feeding it to a parser).

While these environments were tailored for specific grammar formalisms, they share a number of features. Firstly, they are expressive enough to characterize subsets of natural language. Following Shieber (1984), we call this feature *weak completeness*. Secondly, they are *notationally expressive* enough to relatively easily formalize important theoretical notions. Thirdly, they are *rigorous*, that is, the semantics of their underlying language is well defined and understood. Additionally, for an environment to be useful in practice, it should be *simple* to use (by a linguist), and make it possible to detect errors in the described target grammar.

If we consider a particular type of computational grammars, namely tree based grammars, that is grammars where the basic units are trees (or tree descriptions) of arbitrary depth, such as Tree-Adjoining Grammar (TAG, (Joshi, Levy, and Takahashi 1975)), D-Tree Grammar (DTG, (Rambow, Vijay-Shanker, and Weir 1995)), Tree Description Grammars (TDG (Kallmeyer 1999)) or Interaction Grammars (IG, (Perrier 2000)), environments sharing all of the above features are lacking. As we shall see in the Section 7 of this article, there have been some proposals for grammar engineering environments for tree based grammar, for example Candito (1996), Xia, Palmer, and Vijay-Shanker (1999), but these lack notational expressivity. This is partly due to the fact that tree based formalisms offer an extended domain of locality where one can encode constraints between remote syntactic constituents. If one wants to define such constraints while giving a modular and incremental specification of the grammar, one needs a high level of notational expressivity, as we shall see throughout the article (and especially in Section 4).

In this article, we present XMG (eXtensible MetaGrammar), a framework for specifying tree based grammars. Focusing mostly on FB-LTAG (but using IG and MC-TAG to illustrate flexibility), we argue that XMG departs from other existing computational frameworks for designing tree based grammars in three main ways:

- First, XMG is a *declarative* language. In other words, grammaticality is defined in an order-independent fashion by a set of well-formedness constraints rather than by procedures. In particular, XMG permits a fully declarative treatment of diathesis that markedly departs from the procedural rules (called meta- or lexical rules) previously used to specify tree based grammars.
- Second, XMG is *notationally expressive*. The XMG language supports full disjunction and conjunction of grammatical units, a modular treatment of multiple linguistic dimensions, multiple inheritance of units and a sophisticated treatment of identifiers.

We illustrate XMG's notational expressivity by showing (i) how it facilitates the modelling of the tree fragment hierarchies required to specify tree based grammars and (ii) how it permits a natural modelling of the syntax/semantic interface between semantic representations and syntactic trees as can be used in FB-LTAG.

- Third, XMG is *extensible* in that its computational architecture facilitates (i) the integration of an arbitrary number of linguistic dimensions (syntax, semantics, etc.), (ii) the modelling of different grammar formalisms (FB-LTAG, MC-TAG, IG) and (iii) the specification of general linguistic principles (*e.g.*, clitic ordering in French).

The article is structured as follows. Section 2 starts by giving a brief introduction to FB-LTAG, the grammar formalism we used to illustrate most of XMG's features. The next three sections then go on to discuss and illustrate XMG's three main features namely, declarativity, notational expressivity and flexibility. In section 3, we focus on *declarativity* and show how XMG's generalized disjunction permits a declarative encoding of diathesis. We then contrast the XMG approach with the procedural methods previously resorted to for specifying FB-LTAG. Section 4 addresses notational expressivity. We present the syntax of XMG and show how the sophisticated identifier handling it supports permits a natural treatment (i) of identifiers in tree based hierarchies and (ii) of the unification based syntax/semantics interface often used in FB-LTAG. In section 5, we concentrate on extensibility. We first describe the operational semantics of XMG and the architecture of the XMG compiler. We then show how these facilitate the adaptation of the basic XMG language to (i) different grammar formalisms (IG, MC-TAG, FB-LTAG), (ii) the integration of specific linguistic principles such as clitic ordering constraints and (iii) the specification of an arbitrary number of linguistic dimensions. In Section 6, we illustrate the usage of XMG by presenting an XMG specification for the verbal fragment of a large scale FB-LTAG for French augmented with a unification-based semantics. We also briefly describe the various other tree-based grammars implemented using XMG. Section 7 discusses the limitations of other approaches to the formal specification of tree based grammars, while Section 8 concludes with pointers for further research.

2. Tree-Adjoining Grammar

A Tree Adjoining Grammar consists of a set of auxiliary or initial elementary trees and of two tree composition operations namely, substitution and adjunction. Initial trees are trees whose leaves are either substitution nodes (marked with ↓) or terminal symbols (words). Auxiliary trees are distinguished by a foot node (marked with ⋆) whose category must be the same as that of the root node. Substitution inserts a tree onto a substitution node of some other tree while adjunction inserts an auxiliary tree into a tree. Figure 1 shows a toy TAG generating the sentence *Marie a vu Jean* 'Mary has seen John' and sketches its derivation.[1]

Among existing variants of TAG, a commonly used in practice is Lexicalised Feature-Based Tree-Adjoining Grammar (FB-LTAG, (Vijay-Shanker and Joshi 1988)). A lexicalised TAG is such that each elementary tree has at least one leaf labelled with a lexical item (word) while in an FB-LTAG, tree nodes are additionally decorated with two feature structures (called top and bottom). These feature structures are unified during derivation as follows. On substitution, the top features of the substitution node

[1] The elementary trees displayed in this article conform to Abeillé (2002), that is, we reject the use of a VP constituent in French.

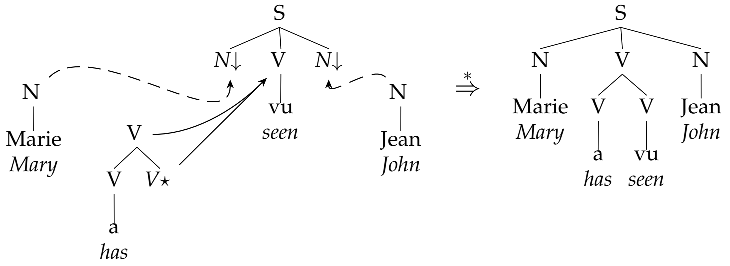


Figure 1
Sample derivation of *Marie a vu Jean* 'Mary has seen John' in a TAG

are unified with the top features of the root node of the tree being substituted in. On adjunction, the top features of the root of the auxiliary tree are unified with the top features of the node where adjunction takes place; and the bottom features of the foot node of the auxiliary tree are unified with the bottom features of the node where adjunction takes place. At the end of a derivation, the top and bottom feature structures of all nodes in the derived tree are unified.

Implementation of Tree-Adjoining Grammars. Most existing implementations of Tree-Adjoining Grammars follow the three-layer architecture adopted for the XTAG grammar (XTAG Research Group 2001), a Feature-Based Lexicalised TAG for English. Thus the grammar consists of (i) a set of so-called tree schemas (*i.e.*, elementary trees having a leaf node labelled with a ⋄ referring to where to anchor lexical items[2]), (ii) a morphological lexicon associating words with lemmas and (iii) a syntactic lexicon associating lemmas with tree schemas (these are gathered into *families* according to syntactic properties, such as the sub-categorization frame for verbs). Figure 2 shows some of the tree schemas associated with transitive verbs in the XTAG grammar. The tree (a) corresponds to a declarative sentence, (b) to a WH-question on the subject, (c) to a passive clause with a BY-agent and (d) to a passive clause with a WH-object. As can be seen, each tree schema contains an anchor node (marked with ⋄). During parsing this anchor node can be replaced by any word morphologically related to a lemma listed in the syntactic lexicon as anchoring the transitive tree family.

This concept of tree family allows us to share structural information (tree schemas) between words having common syntactic properties (*e.g.*, sub-categorization frames). There still remains a large redundancy within the grammar since many elementary tree schemas share common subtrees (large coverage Tree Adjoining Grammars usually consist of hundreds, sometimes thousands, of tree schemas). An important issue when specifying such grammars is thus structure sharing. Being able to share structural information is necessary not only for a faster grammar development, but also for an easier grammar maintenance (modifications to be applied to the tree schemas would be restricted to shared structures). In the next Section, we will see how XMG declarativity can be efficiently used to factorize Tree Adjoining Grammars. In addition, Section 4 will show how XMG notational expressivity facilitates the specification of another commonly used tree sharing device, namely inheritance hierarchies of tree fragments.

[2] As mentioned above, we describe lexicalised TAG, thus every tree schema has to contain at least one anchor (node labelled ⋄).

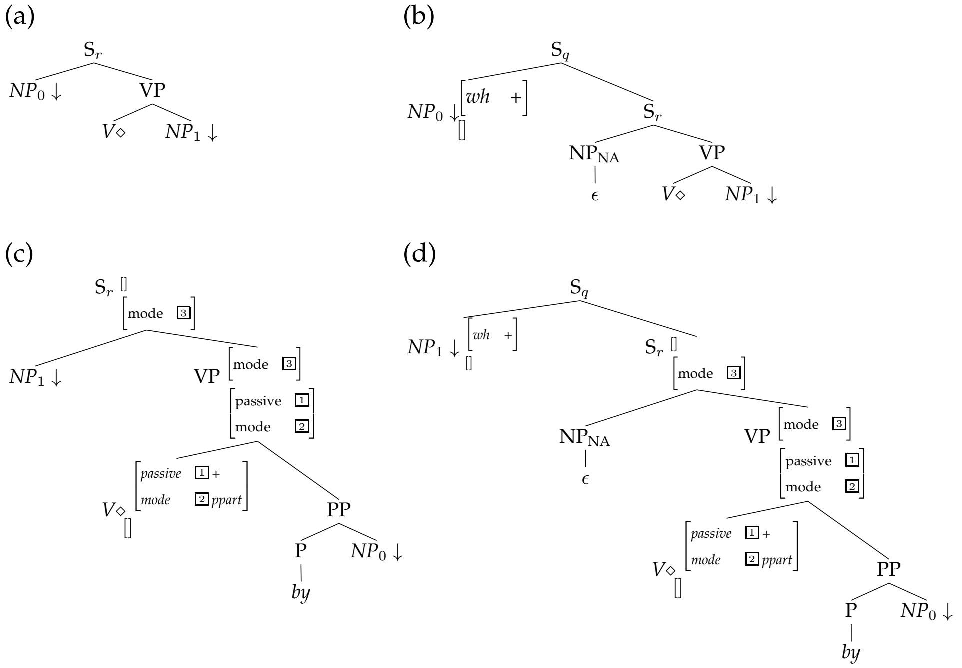
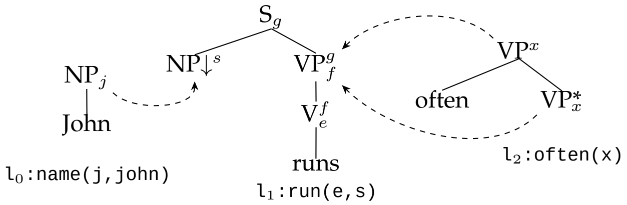


Figure 2
Some tree schemas for English transitive verbs

Extending TAG with a unification-based semantics. To extend FB-LTAG with a compositional semantics, Gardent and Kallmeyer (2003) propose to associate each elementary tree with a flat semantic representation. For instance, in Figure 2, the trees[3] for *John, runs* and *often* are associated with the semantics l_0:name(j,john), l_1:run(e,s) and l_2:often(x) respectively. Importantly, the arguments of semantic functors are represented by unification variables which occur both in the semantic representation of this functor and on some nodes of the associated syntactic tree. Thus in Figure 2, the semantic index s occurring in the semantic representation of *runs* also occurs on the subject substitution node of the associated elementary tree. The value of semantic arguments is then determined by the unifications resulting from adjunction and substitution. For instance, the semantic index s in the tree for *runs* is unified during substitution with the semantic index j labelling the root node of the tree for *John*. As a result, the semantics of *John often runs* is {l_0:name(j,john), l_1:run(e,j), l_2:often(e)}.

Gardent and Kallmeyer (2003)'s proposal was applied to various semantic phenomena (Kallmeyer and Romero 2004a, 2004b, 2008). Its implementation however relies on having a computational framework which associates syntactic trees with flat semantic formulae while allowing for shared variables between trees and formulae. In the following sections, we will show how XMG notational expressivity makes it possible to specify an FB-LTAG equipped with a unification based semantics.

[3] C^x/C_x abbreviate a node with category C and a top/bottom feature structure including the feature-value pair { **index :** x}.



⇒ l_0:name(j,john), l_1:run(e,j), l_2:often(e)

Figure 3
A toy lexicalised FTAG with unification-based semantics (l_0, l_1, l_2, e and j are constants while s, f, g, x are unification variables).

3. Declarativity

In this section, we show how a phenomenon which is often handled in a procedural way by existing approaches, can be provided with a declarative specification in XMG. Concretely, we show how XMG supports a declarative account of diathesis that avoids the drawbacks of lexical rules (*e.g.*, information erasing). We start by presenting the lexical rule approach. We then contrast it with the XMG account.

3.1 Capturing diathesis using lexical rules

Following Flickinger (1987), redundancy among grammatical descriptions is often handled using two devices: an inheritance hierarchy and a set of lexical rules. While the inheritance hierarchy permits us to encode the sharing of common substructures, lexical rules (sometimes called meta-rules) permits us to capture relationships between trees by deriving new trees from already specified ones. For instance, passive trees will be derived from active ones.

Although Flickinger (1987)'s approach was developed for Head-Driven Phrase Structure Grammar (HPSG), several similar approaches have been put forward for FB-LTAG (Vijay-Shanker and Schabes 1992; Becker 1993; Evans, Gazdar, and Weir 1995; XTAG Research Group 2001). One important drawback of these approaches however, is that they are procedural in that the order in which lexical rules apply matters. For instance, consider again the set of trees given in Figure 2. In the meta-rule representation scheme adopted by Becker (1993), the base tree (a) would be specified in the inheritance hierarchy grouping all base trees, and the derived trees (b,c,d) would be generated by applying one or more meta-rules on this base tree. Figure 4 sketches these meta-rules. The left hand side of the meta-rule is a matching pattern replaced with the right hand side of the meta-rule in the newly generated tree. Symbol "?" denotes a meta-variable whose matching subtree in the input is substituted in place of the variable in the output tree. Given these, the tree family in Figure 2 is generated as follows: (b) and (c) are generated by application to the base tree (a) of the *Wh-Subject* and *Passive* meta-rules respectively. Further, (d) is generated by applying first, the Wh-Subject meta-rule and second, the Passive meta-rule to the base tree.

More generally a meta-rule is a procedural device that given a tree instance, generates a new tree instance by adding, suppressing (hence possibly substituting) information in grammatical units. Prolo (2002) defines a set of meta-rules which can be used to specify a large FB-LTAG for English. Given an ordered set of meta-rules however,

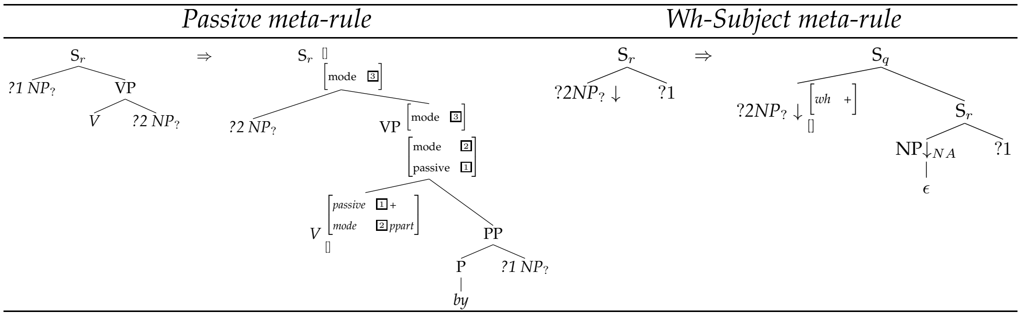


Figure 4
Simplified meta-rules for passive and wh-subject extraction

there is no guarantee that the trees they derive are linguistically appropriate and that the derivation process terminates. Thus to ensure termination and consistency, Prolo (2002) needs to additionally provide rule ordering schemes (expressed as automata).

3.2 XMG: capturing diathesis using disjunction

XMG provides an alternative account for describing tree sets such as that of Figure 2 without lexical rules and without the related ordering constraints. In essence, the approach consists in enumerating trees by combining tree fragments using conjunction and disjunction. More specifically, the tree set given in Figure 2 can be generated by combining some of the tree fragments sketched in Figure 5 using the following conjunctions and disjunctions:[4]

$$\mathit{Subject} \rightarrow \mathit{CanonicalSubject} \vee \mathit{Wh\text{-}NP\text{-}Subject} \quad (1)$$

$$\mathit{ActiveTransitiveVerb} \rightarrow \mathit{Subject} \wedge \mathit{ActiveVerb} \wedge \mathit{CanonicalObject} \quad (2)$$

$$\mathit{PassiveTransitiveVerb} \rightarrow \mathit{Subject} \wedge \mathit{PassiveVerb} \wedge \mathit{CanonicalByObject} \quad (3)$$

$$\mathit{TransitiveVerb} \rightarrow \mathit{ActiveTransitiveVerb} \vee \mathit{PassiveTransitiveVerb} \quad (4)$$

The first clause (*Subject*) groups together two subtrees representing the possible realizations of a subject (canonical and wh). The next two clauses define a tree set for active and passive transitive verbs respectively. The last clause defines the *TransitiveVerb* family as a disjunction of the two verb forms (passive or active). In sum, the *TransitiveVerb* clause defines the tree set sketched in Figure 2 as a disjunction of conjunctions of tree fragments.

One of the issues of meta-rules reported by Prolo (2002) is the handling of feature equations. For a number of cases (including subject relativization in passive trees), ad-hoc meta-rules are needed, for a unified tree transformation cannot be defined. In a declarative approach such as the one above, dealing with feature equations can be done relatively easily. Let us imagine we now want to extend the trees of Figure 2 with feature

[4] For now, let us consider that the tree fragments are combined in order to produce minimal trees by merging nodes whose categories (and features) unify. In the next section, we will see how to precisely control node identification using either node variables or node constraints.

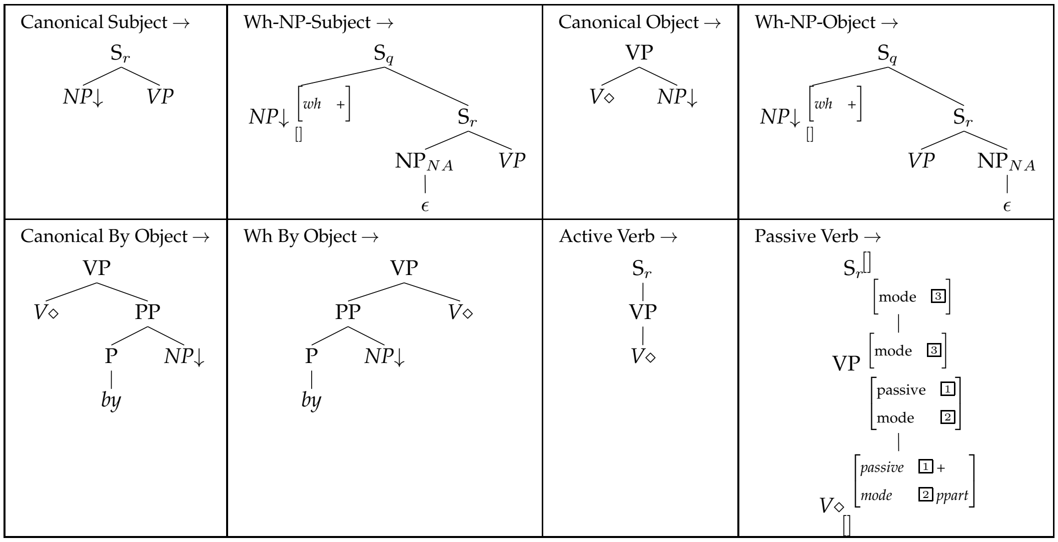
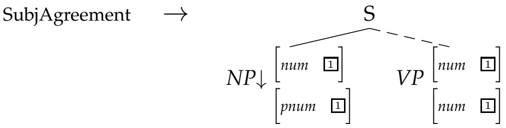


Figure 5
Tree fragments

equations for subject number agreement. We can for instance do so by defining the tree fragment below (the dashed line indicates that the *VP* node can be a descendant, not only a daughter, of the *S* node):[5]



Then we extend the definition of *Subject* as follows:

$$Subject \quad \rightarrow \quad SubjAgreement \wedge (\ CanonicalSubject \vee Wh\text{-}NP\text{-}Subject\ ) \qquad (5)$$

If we want to get further with the description of transitive verbs, for instance by taking into account wh- objects and by-objects, this can be done as follows. We first define the elementary fragments *Wh-NP-Object* and *Wh-By-Object* (see Figure 5), and then define the following additional combinations:[6]

$$ActiveTransitiveVerb \quad \rightarrow \quad CanonicalSubject \wedge ActiveVerb \wedge Wh\text{-}Np\text{-}Object \qquad (6)$$

$$PassiveTransitiveVerb \quad \rightarrow \quad CanonicalSubject \wedge PassiveVerb \wedge Wh\text{-}By\text{-}Object \qquad (7)$$

[5] Note that in XMG, it is not mandatory to define any tree structure inside *SubjAgreement*. We could define independent NP and VP nodes, and associate them with variables, say n_1 and n_2. n_1 and n_2 would then be exported and reused directly in the classes *CanonicalSubject* and *Wh-NP-Subject* respectively.

[6] Note that these clauses only consider canonical subjects to avoid having both a Wh-subject and a Wh-object. This is not entirely satisfactory, as we would prefer to define a single abstraction over objects (as was done for subjects) and use it wherever possible. There would then be another mechanism to capture this exception and cause the invalid combination to fail (that is, the resulting tree description not to have any model). Such a mechanism exists in XMG, and is called linguistic principle, see Section 5.

Evans *et al.* (1995) argue for the necessity of using lexical rules for grammatical description based on two arguments: (i) morphology is irregular and has to be handled by a non monotonic device and (ii) erasing rules such as the agentless passive (*John eats an apple / An apple is eaten*) are needed to erase an argument from the canonical base tree. Neither of these arguments holds here however: the first argument because we describe tree schema hence lexical and morphological issues are ruled out; the second because agentless passive and more generally, argument erasing constructions, can simply be defined by an additional clause such as:

$$\mathit{AgentlessPassiveTransitiveVerb} \quad \rightarrow \quad \mathit{Subject} \wedge \mathit{PassiveVerb} \tag{8}$$

To sum up, using a declarative language to specify a tree based grammar offer an adequate level of control on the structures being described while avoiding to deal with ordering and termination issues. It facilitates grammar design and maintenance, by providing an abstract view on grammar trees, uniquely made of monotonic (no information removal) combinations of tree fragments.

4. Notational expressivity

We now focus on notational expressivity and show how XMG supports a direct encoding of (i) distinct linguistic dimensions (here syntax, semantics and the syntax/semantics interface) and (ii) the various types of coreferences[7] which arise in the development of tree based grammars.

The syntax of the XMG language can be formally defined as follows.

$$\mathit{Class} \quad ::= \quad \mathit{Name}^{C_1,\ldots,C_k}_{x_1,\ldots,x_n} \quad \rightarrow \quad \mathit{Content} \tag{9}$$

$$\mathit{Content} \quad ::= \quad \langle \mathit{SYN, SEM, DYN} \rangle \mid \mathit{Name} \mid \mathit{Content} \wedge \mathit{Content} \mid \mathit{Content} \vee \mathit{Content} \tag{10}$$

$$\begin{aligned} \mathit{SYN} \quad ::= \quad & n_1 \rightarrow n_2 \mid n_1 \rightarrow^+ n_2 \mid n_1 \rightarrow^* n_2 \mid n_1 \prec n_2 \mid n_1 \prec^+ n_2 \mid n_1 \prec^* n_2 \mid \\ & n_1[f_1 : v_1, \ldots , f_k : v_k] \mid n_1(c_1 : cv_1, \ldots , c_l : cv_l) \mid n_1 = n_2 \mid x = C_i.y \mid \\ & n_1\,(c_1 : cv_1, \ldots , c_l : cv_l)\,[f_1 : v_1, \ldots , f_k : v_k] \mid \mathit{SYN} \wedge \mathit{SYN} \end{aligned} \tag{11}$$

$$\mathit{SEM} \quad ::= \quad l_i : p(E_1, \ldots, E_n) \mid l_i \leq h_j \mid \mathit{SEM} \wedge \mathit{SEM} \tag{12}$$

$$\mathit{DYN} \quad ::= \quad \langle f_1 : v_1 , \ldots , f_n : v_n \rangle \tag{13}$$

Here and in what follows, we use the following notational conventions. C_i denote variables over class names, x_i, x and y are variables ranging over tree nodes or feature values, n_i refer to node variables, f, f_i are features and v, v_i, feature values (constants or variables), l_i, h_j, p and E_i are variables over semantic labels, semantic holes, predicates and predicate arguments in flat semantic formulae respectively.[8] [ ] are used to associate a node variable with some feature constraint. ( ) are used to associate a node variable with some property constraint (*e.g.*, node colors, see Section 5). c_i and cv_i denote a

[7] By coreference, we mean the sharing of information between distinct elementary fragments of the grammar specification.

[8] See (Gardent and Kallmeyer 2003) for a detailed introduction to flat semantics.

property constraint and a property constraint value respectively. $C_1.y$ denotes the y variable declared in class C_i and $=$ is unification; $\prec$ and $\rightarrow$ denote linear precedence and immediate dominance relations between nodes. Finally, $+, *$ represent the transitive and transitive-reflexive closure of a relation respectively.

The first two clauses of the formal definition above specify XMG classes and how they combine. The next three clauses define the languages supported for describing three linguistic dimensions, namely syntax (SYN), semantics (SEM) and the syntax/semantics interface (called DYN for dynamic interface). We now discuss each of these in more details starting bottom-up with the three linguistic dimensions and ending with the control language that permits us to combine basic linguistic units into bigger ones.

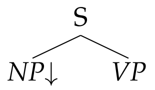
SYN. The XMG formalism for syntax (copied below for convenience) is a tree description logic similar to that proposed by Vijay-Shanker and Schabes (1992), Rogers and Vijay-Shanker (1994) to describe tree based grammars.

$$\begin{array}{lll} \mathit{SYN} & ::= & n_1 \rightarrow n_2 \;\mid\; n_1 \rightarrow^{+} n_2 \;\mid\; n_1 \rightarrow^{*} n_2 \;\mid\; n_1 \prec n_2 \;\mid\; n_1 \prec^{+} n_2 \;\mid\; n_1 \prec^{*} n_2 \;\mid \\ & & n_1[f_1 : v_1, \ldots, f_k : v_k] \;\mid\; n_1(c_1 : cv_1, \ldots, c_l : cv_l) \;\mid\; n_1 = n_2 \;\mid\; x = C_i.y \;\mid \\ & & n_1\,(c_1 : cv_1, \ldots, c_l : cv_l)\,[f_1 : v_1, \ldots, f_k : v_k] \;\mid\; \mathit{SYN} \,\wedge\, \mathit{SYN} \end{array}$$

It includes tree node variables, feature names, feature values and feature variables. Tree node variables can be related by equality (node identification), precedence (immediate or non immediate) and dominance (immediate or non immediate). Tree nodes can also be labelled with feature structures of depth 2, that is, sets of feature/value pairs where feature values are either variables, constants (*e.g.*, syntactic category), or non recursive feature structure (*e.g.*, top and bot feature structures).

Here is a graphical illustration of how tree logic formulae can be used to describe tree fragments: the depicted tree fragment is a model satisfying the given formula.

$$\begin{array}{llllll} & n_1 \rightarrow n_2 & \wedge & n_1 \rightarrow n_3 & \wedge & n_2 \prec n_3 \\ \wedge & n_1[cat : S] & \wedge & n_2(mark : subst)\,[cat : NP] & \wedge & n_3[cat : VP] \end{array}$$



One distinguishing feature of the XMG tree language is the introduction of node constraints ($n_1(c : cv)$) which generalize Muskens and Krahmer's (1998) use of positive and negative node markings. Concretely, node constraints are attribute-value matrices, which contain information to be used when solving tree descriptions to produce grammar trees. In other words, node constraints are used to further restrict the set of models satisfying a tree description. As an example of node constraint, consider node annotations in FB-LTAG (foot node, substitution node, null-adjunction, etc.). Such annotations can be used as node constraints to allow the description solver to apply well-formedness constraints (*e.g.*, there is at most one foot node).

Another interesting feature of XMG concerns the inclusion of the dot operator which permits us to identify variables across classes in cases where name sharing cannot be resorted to. When a variable y is declared in a class C, the latter being instantiated within a class D, y can be accessed from D by $C.y$ (the identifier y still being available in D's namespace).

SEM. The semantic dimension supports a direct encoding of the flat semantic formulae used by Gardent and Kallmeyer (2003):

$$\mathit{SEM} \quad ::= \quad l_i : p(E_1, ..., E_n) \ \mid \ l_i \leq h_j \mid \ \mathit{SEM} \ \wedge \ \mathit{SEM}$$

where $l_i : p(E_1, ..., E_n)$ represents a predicate p with label l_i and arguments $E_1, ..., E_n$ whilst $l_i \leq h_j$ is a scope constraint between label l_i and scope h_j. Expressions (predicate arguments E_i) can refer to semantic holes, constants (atomic values) or unification variables (written x, y below).

For instance, the following flat semantic formula can be used to underspecify the meaning of the sentence “Every dog chases a cat”:

$$\begin{gathered} l_0 : \forall(x, h_1, h_2) \quad \wedge \quad l_1 \leq h_1 \quad \wedge \quad l_1 : \text{Dog}(x) \quad \wedge \quad l_2 \leq h_2 \quad \wedge \quad l_2 : \text{Chase}(x, y) \\ \wedge \quad l_3 : \exists(y, h_3, h_4) \quad \wedge \quad l_4 \leq h_3 \quad \wedge \quad l_4 : \text{Cat}(y) \quad \wedge \quad l_2 \leq h_4 \end{gathered} \tag{14}$$

This formula denotes the two first order logic formulae below thereby describing the two possibles readings of this sentence[9].

$$l_0 : \forall(x, l_1, l_3) \ \wedge \ l_1 : \text{Dog}(x) \ \wedge \ l_2 : \text{Chase}(x, y) \ \wedge \ l_3 : \exists(y, l_4, l_2) \ \wedge \ l_4 : \text{Cat}(y) \tag{15}$$

$$l_0 : \forall(x, l_1, l_2) \ \wedge \ l_1 : \text{Dog}(x) \ \wedge \ l_2 : \text{Chase}(x, y) \ \wedge \ l_3 : \exists(y, l_4, l_0) \ \wedge \ l_4 : \text{Cat}(y) \tag{16}$$

DYN. The *DYN* dimension generalizes Kinyon’s *hypertag* (Kinyon 2000) which are unified whenever two tree fragments are combined. Similarly, in XMG the *DYN* dimension is a feature structure that is unified whenever two XMG classes are combined through inheritance or through conjunction (see below, the discussion on XMG control language).

For instance, the following constraints ensure a coreference between the index I occurring in the syntactic dimension and the argument X occurring in the semantic dimension ($index_{subject}$ and arg_1 are feature names, and E, I, X and V local unification variables).

$$C_1 \quad \rightarrow \quad Node\,[idx : I] \ \wedge \ \langle index_{subject} : I \rangle \tag{17}$$

$$C_2 \quad \rightarrow \quad L : P(E) \wedge L : Theta_1(E, X) \ \wedge \ \langle arg_1 : X \rangle \tag{18}$$

$$SubjectArg_1 \quad \rightarrow \quad C_1 \quad \wedge \quad C_2 \quad \wedge \ \langle index_{subject} : V, arg_1 : V \rangle \tag{19}$$

More generally, the *DYN* dimension permits us to unify nodes and feature values that belong to distinct classes and dimensions, and are thus often not related within the inheritance hierarchy. As we shall see in Section 6, the *DYN* dimension permits a modular account of the syntax/semantic interface in which linking constraints can be stipulated separately and reused to specify the various diatheses.

In other words, the *DYN* feature structure allows us to extend the scope of some specific variables, so that they can be unified with variables (or values) introduced

[9] For more details on the interpretation of flat semantics and on its association with a grammar of natural language, see (Gardent 2008).

in some other classes of the metagrammar. This concept of scope extension can be compared with that of *hook* in Copestake, Lascarides, and Flickinger (2001).

Control language. The linguistic units (named *Content* here) defined by the linguist can be abstracted and combined as follows:

$$\begin{array}{rcl} \mathit{Class} & ::= & \mathit{Name}^{C_1,\ldots,C_k}_{x_1,\ldots,x_n} \;\; \rightarrow \;\; \mathit{Content} \\ \mathit{Content} & ::= & \langle \mathit{SYN}, \mathit{SEM}, \mathit{DYN} \rangle \mid \mathit{Name} \mid \mathit{Content} \wedge \mathit{Content} \mid \mathit{Content} \vee \mathit{Content} \end{array}$$

The first clause states that the linguistic information encoded in *Content* is abstracted in a class named $Name$ and that this class inherits classes $C_1, \ldots, C_k$ and exports variables $x_1, \ldots, x_n$. That is, XMG allows for abstraction, inheritance and variable exports. By default, variables (referring to nodes and feature values) are local to a class. However, export statements extend the scope of a variable to all sub classes. An exported variable can also be accessed from outside its class in case of class instantiation (using the dot operator introduced earlier in this section). The second clause states that an XMG class consists of a syntactic, a semantic and a dynamic description (each of them possibly empty): and that XMG classes can be combined by conjunction and disjunction and reused through class instantiation. The notation $\langle \mathit{SYN}, \mathit{SEM}, \mathit{DYN} \rangle$ represents simultaneous contributions (possibly empty) to all 3 dimensions.[10]

The XMG control language differs from other frameworks used to specify tree based grammars (Vijay-Shanker and Schabes 1992; Xia et al. 1998; Candito 1999) in two main ways. First, it supports generalized conjunctions and disjunctions of classes. As shown in Section 3, this permits *inter alia* a declarative treatment of diathesis.

Second, it allows for both local and exported variables. As mentioned in Section 3, a common way to share structure within a tree-based grammar is to define an inheritance hierarchy of either tree fragments (Evans, Gazdar, and Weir 1995) or tree descriptions (Vijay-Shanker and Schabes 1992; Candito 1996; Xia 2001). When considering an FB-LTAG augmented with unification semantics, the hierarchy will additionally contain semantic representations and/or tuples made of tree fragments and semantic representations. In all cases, the question arises of how to handle identifiers across classes and more specifically, how to share them.

In Candito's approach (1996), tree nodes are referred to using constants so that multiple occurrences of the same node constant refer to the same node. As pointed out in Gardent and Parmentier (2006), global names have several non trivial shortcomings. First, they complicate grammar writing in that the grammar writer must remember the names used and their intended interpretation. Second, they fail to support multiple uses of the same class within one class. For instance, in French, some verbs sub-categorize for two prepositional phrases (PP). A natural way of deriving the tree for such verbs would be to combine a verbal tree fragment with two instances of a PP fragment. If however, the nodes in the PP fragment are labelled with global names, then the two occurrences of these nodes will be identified thereby blocking the production of the appropriate tree.[11]

A less restrictive treatment of identifiers is proposed by Vijay-Shanker and Schabes (1992) where each tree description can be associated with a set of declared node variables and subsets of these node variables can be referred to by descriptions in the

[10] While formally precise, this notation can be cumbersome. In the interest of legibility we adopt throughout the convention that *SYN* stands for $\langle \mathit{SYN}, \, , \, \rangle$, *SEM* for $\langle \, , \mathit{SEM}, \, \rangle$, ad *DYN* for $\langle \, , \, , \mathit{DYN} \rangle$.

[11] An analogous situation may arise in English with ditransitive verbs requiring two direct objects.

hierarchy that inherit from the description in which these node variables were declared. For instance, if entity A in the hierarchy declares such a special node variable X and B inherits from A, then X can be referred to in B using the notation $A.X$.[12]

XMG generalizes Vijay-Shanker and Schabes' approach (1992) by integrating an export mechanism that can be used to extend the scope of a given identifier (node or feature value variable) to classes that inherit from the exporting class. Thus if class B inherits from class A and class A exports variable X, then X is visible in B and its reuse forces identity. If B inherits from several classes and two (or more) of these inherited classes export the *same* variable name X, then X is not directly visible from B. It can be accessed though using the dot operator. First A is identified with a local variable (*e.g.*, $T = A$), then $T.X$ can be used to refer to the variable X exported by A.

To sum up, XMG allows for local variables to be exported to sub-classes as well as for prefixed variables *i.e.*, variables that are prefixed (using the dot operator) with a reference to the class in which they are declared. In this way, the pitfalls introduced by global names are avoided while providing enough expressivity to handle variable coreference (via the definition of variable namespaces). Section 6 will further illustrate the use of the various coreference devices made available by XMG showing how they concretely facilitate grammar writing.

Let us finally illustrate variable handling with XMG on the example of Figure 2. Recall we define the trees of Figure 2 as the conjunctions and disjunctions of some tree fragments of Figure 5, such as:

$$\textit{Subject} \quad \rightarrow \quad \textit{SubjAgreement} \wedge (\ \textit{CanonicalSubject} \vee \textit{Wh-NP-Subject}\ ) \tag{20}$$

CanonicalSubject can be defined as a tree description formula as follows (only variables n_2 and n_3 are exported):

$$\textit{CanonicalSubject}_{n_2,n_3} \quad \rightarrow \quad \begin{array}{l} n_1 \rightarrow n_2 \wedge n_1[cat : S] \wedge n_2(mark : subst)\,[cat : NP] \wedge \\ n_1 \rightarrow n_3 \wedge n_3[cat : VP] \wedge n_2 \prec n_3 \end{array} \tag{21}$$

The class *Wh-NP-Subject* is defined accordingly (*i.e.*, by means of a slightly more complex tree description formula using the n_2 and n_3 variable identifiers to refer to the nodes involved in subject agreement). The class *SubjAgreement* is defined slightly differently (we do not impose any tree relation between the node concerned with number agreement):

$$\textit{SubjAgreement}_{n_1,n_2} \quad \rightarrow \quad \begin{array}{l} n_1\,[[top : [num : x]]\,[bot : [num : x]]] \wedge \\ n_2\,[[top : [num : x]]\,[bot : [num : x]]] \end{array} \tag{22}$$

[12] In fact, the notation used by Vijay-Shanker and Schabes (1992) is *attr:X* with attr an attribute variable ranging over a finite set of attributes, to indicate special node variables that scope outside their class; and *attr(A)* to refer to such variables from outside the entity in which they were declared. We use a different notation here to enforce consistency with the XMG notation.

We can then explicitly control the way the fragments combine as follows:

$$\textit{Subject} \quad \rightarrow \quad \begin{array}{l} C_1 = \textit{SubjAgreement}_{n_1,n_2} \wedge \\ C_2 = (\ \textit{CanonicalSubject}_{n_2,n_3} \vee \textit{Wh-NP-Subject}_{n_2,n_3}\ ) \wedge \\ C_1.n_1 = C_2.n_2 \ \wedge \ C_1.n_2 = C_2.n_3 \end{array} \tag{23}$$

In this example, we see how to constrain, via variable export and unification, some given syntactic nodes to be labelled with feature structures defined somewhere else in the metagrammar. We use XMG's flexible management of variable scope to deal with node coreference. Compared with previous approaches on metagrammars such as (Candito 1996; Xia 2001), having the possibility to handle neither only global nor only local variables offers a high level of expressivity along with a precise control on the structures being described.

5. Extensibility

A third distinguishing feature of XMG is extensibility. XMG is extensible in that (i) dimensions can be added and (ii) each dimension can be associated with its own interpreter. In order to support an arbitrary number of dimensions, XMG relies on a device permitting the accumulation of an arbitrary number of types of literals, namely *Extensible Definite Clause Grammar* (EDCG) (Van Roy 1990). Once literals are accumulated according to their type (*i.e.*, each type of literals is accumulated separately), they can be fed to dedicated interpreters. Since each of these sets of literals represents formulas of a description language, these interpreters are solvers, whose role is to compute models satisfying the accumulated formulas.

Via this concept of separated dimensions, XMG allows us (i) to describe different levels of language (not only syntax, but also semantics and potentially morphology,[13] etc.), and (ii) to define *linguistic principles* (well-formedness constraints to be applied on the structures being described). These principles depend either on the dimension (*e.g.*, scope constraints in flat semantics), the target formalism (*e.g.* cooccurrence predicate-arguments in FB-LTAG) or the natural language (*e.g.*, clitic ordering in Romance languages) being described.

In what follows, we start by showing how XMG handles dimensions independently from each other introducing EDCG (Section 5.1). We then summarize the architecture of the XMG system (Section 5.2). We finally show how different solvers can be used to implement various constraints on each of these dimensions (Section 5.3). In particular, we discuss three kinds of extensions implemented in XMG: extension to several grammar formalisms, integration of explicit linguistic generalizations and inclusion of color-based node marking to facilitate grammar writing.

5.1 XMG: accumulating and interpreting an arbitrary number of descriptions

Accumulating (tree) descriptions. First, let us notice that XMG is nothing else than a logic language *à la* Prolog (Duchier, Parmentier, and Petitjean 2012). More precisely, an XMG

[13] Recently, XMG has been used to describe the morphology of verbs in Ikota, a Bantu language spoken in Gabon (Duchier et al. 2012).

specification is a collection of Horn clauses, which contribute a declarative description of what a computational tree grammar is.

Logic Program			XMG Metagrammar		
$Clause$	$::=$	$Head \rightarrow Body$	$Class$	$::=$	$Name \rightarrow Content$
$Body$	$::=$	$Fact \mid Head \mid$	$Content$	$::=$	$Description \mid Name \mid$
		$Body \vee Body \mid$			$Content \vee Content \mid$
		$Body \wedge Body$			$Content \wedge Content$
$Query$	$::=$	$Head$	$Axiom$	$::=$	$Name$

Recall that the descriptions handled by XMG are in fact tuples of the form $\langle SYN, SEM, DYN\rangle$. An XMG class can thus describe, in a non-exclusive way, any of these 3 levels of description. If one wants to add another level of description (*i.e.*, another dimension), one needs to extend the arity of this tuple. Before discussing this, let us first see how such tuples are processed by XMG.

As mentioned above, XMG's control language is comparable to Horn clauses. A common way to represent Horn clauses is by using *Definite Clause Grammar* (DCG) (Pereira and Warren 1980). Concretely, a DCG is a rewriting system (namely a context-free grammar), where the symbols of the rewriting rules are equipped with pairs of unification variables (these are usually called *difference list* or *accumulator*) (Blackburn, Bos, and Striegnitz 2006, page 100). As an illustration, consider the toy example below.

```
s   --> np,vp.     np  --> det,n.
vp  --> v,np.      vp  --> v.
det --> [the].     det --> [a].
n   --> [cat].     n   --> [mouse].
v   --> [eats].
```

The string language described by this DCG can be obtained by submitting the query `s(X,[])` where `X` is a unification variable to be bound with lists of facts (these being the sentences belonging to the string language). As we can easily see, this language contains the sentences “a cat eats”, “the cat eats”, “a mouse eats”, “the mouse eats”, “a cat eats a mouse”, “a mouse eats a cat”, etc.

Similarly, we can represent XMG classes as DCG clauses. For instance, the combinations of syntactic fragments given on page 8 can be rewritten as DCG clauses as follows:

```
subject                --> canonicalSubject.
subject                --> whNpSubject.
activeTransitiveVerb   --> subject, activeVerb, canonicalObject.
passiveTransitiveVerb  --> subject, passiveVerb, canonicalByObject.
transitiveVerb         --> activeTransitiveVerb.
transitiveVerb         --> passiveTransitiveVerb.
```

Disjunctions (*e.g.*, the *subject* specification) translate to multiple clauses with identical heads and conjunctions (*e.g.*, *activeTransitiveVerb*) to a clause body.

In our case, the terminal symbols of the underlying DCG are not just facts, but tuples of descriptions. In other words, the DCG clause whose head is `canonicalSubject` is associated with a tuple of the following form (the dots have to be replaced with adequate descriptions, these can contain unification variables, whose scope is by default local to the clause):

```
canonicalSubject  --> [desc(syn(...),sem(...),dyn(...))].
```

In order to allow for an extension of XMG to an arbitrary number of dimensions, instead of compiling XMG classes into a DCG whose accumulator stores tuples with a fixed arity, these classes are compiled into an EDCG (Van Roy 1990). EDCG are DCG with multiple accumulators. In XMG, each dimension is thus allocated a dedicated accumulator in the underlying EDCG.

Note that although the content of the various dimensions is accumulated separately, dimensions may nevertheless share information either via local unification variables (if the XMG class defines several dimensions locally), or via exported unification variables (in case of class instantiation or inheritance), or via the shared unification variables supported by the *DYN* dimension.

At the end of the EDCG execution, we obtain, for each axiom of the metagrammar (*i.e.*, for each class name to be valuated), a list of description formulas per accumulator. These lists are grouped together into a tuple of lists of the following form (*N* is the number of dimensions, and consequently of accumulators):

```
desc(accu1(L1),accu2(L2), ... ,accuN(LN))
```

Each element (*i.e.*, list *Li*) of such a tuple is a complete description of a given dimension, where shared variables have been unified (via unification with backtracking).

Solving (tree) descriptions. As illustrated above, interpreting XMG's control language in terms of an EDCG yields tuples whose arity is the number of dimensions defined by the linguist, that is, triples of the form $\langle SYN, SEM, DYN \rangle$ if syntax, semantics and the dynamic interface are described.

For each dimension D, XMG includes a constraint solver S_D which computes the set of minimal models $M_D = S_D(d_D)$ satisfying the description (d_D) of that dimension. In other words, each dimension is interpreted separately by a specific solver. For instance, the syntactic dimension is handled by a tree description solver which produces, for a given tree description, the set of trees satisfying that description, while the solver for the semantic dimension simply outputs the flat semantic representation (list of semantic literals) built by the EDCG through accumulation.

Note that, while solvers are distinct, the models computed in each dimension may nonetheless be coupled through shared variables. In that case, these variables can constrain the models computed by the respective solvers. For instance, shared variables can be used for the syntactic tree description solver to be parametrized by some value coming from the semantic input description. Note that the output of the solving process is a Cartesian product of the sets of minimal models of each solver. As a consequence, the worst case complexity of metagrammar compilation is that of the various solvers associated with relevant dimensions.

In addition to having separate solvers for each dimension, the constraint solving approach used in XMG permits us to modularize a given solver by combining different *principles*. Each such principle enforces specific constraints on the models satisfying the description of a given dimension. For instance, for the syntactic dimension of an FB-LTAG, a set of principles is used to enforce that the structures produced by the compiler are trees, and that these conform to the FB-LTAG formalism (*e.g.*, there is no tree having two foot nodes).

5.2 Architecture

The XMG compiler[14] consists of the following three modules:

[14] The XMG compiler is open source software released under the terms of the CeCILL GPL-compliant licence. See http://sourcesup.renater.fr/xmg.

- A compiler that parses XMG's concrete syntax and compiles XMG classes into clauses of an Extended Definite Clause Grammar (EDCG).
- A virtual machine (VM), which interprets EDCG. This VM performs the accumulation of dimensions along with scope management and identifiers resolution. This VM is basically a unification engine equipped with backtracking, and which is extended to support EDCG. While its architecture is inspired by the Warren Abstract Machine (Aït-Kaci 1991), it uses structure sharing to represent and unify prolog terms, and given a query on a class, processes the conjunctions, disjunctions, inheritance and export statements related to that class to produce its full definition namely, a tree description for the *SYN* dimension, a flat semantic formula for the *SEM* dimension and a feature structure for the *DYN* dimension.
- A constraint solving phase which produces for each dimension, the minimal models satisfying the input description as unfolded by the preceding two steps.

As already mentioned, the first part is extensible in that new linguistic dimensions can be added by specifying additional dedicated accumulators to the underlying EDCG. The second part is a unification engine that interprets EDCG while performing both term unification and polarized unification (*i.e.*, unification of polarized feature structures, as defined by Perrier (2000), and discussed in Section 5.3.1). This extended unification is the reason why XMG does not merely recourse to an existing Prolog engine to process EDCG, but relies on a specific VM instead.

The third part is completely modular in that various constraint solvers can be plugged in depending on the requirements set by the dimensions used, and the chosen grammatical framework. For instance, the *SYN* dimension is solved in terms of tree models, while the *SEM* dimension is solved in terms of underspecified flat semantic formulae (*i.e.*, the input semantics remains untouched modulo the unification of its shared variables).

Importantly, these additional solvers can be "turned on/off" (via a primitive of the XMG language) so that for instance, the same processor can be used to compile an XMG specification for an FB-LTAG using linguistic principles such as those defined in the next section (*i.e.*, clitic ordering principle) or not.

5.3 Three extensions of XMG

We now show (i) how the modular architecture of the XMG compiler permits us to specify grammars for several tree based linguistic formalisms; (ii) how it can be extended to enforce language specific constraints on the syntactic trees; and (iii) how additional formal constraints (namely node marking) can be integrated to simplify node identifications (and consequently grammar writing).

5.3.1 TAG**,** MC-TAG **and** IG**: producing trees, tree sets or tree descriptions.** XMG integrates a generic tree solver which computes minimal tree models from a tree description logic formulae built on the language *SYN* introduced in Section 4. This solver integrates the dominance solving technique proposed by Duchier and Niehren (2000) and can be summarized as follows. A minimal tree model is described in terms of the relative positions of its nodes. For each node n in a minimal tree model T, the set of all the nodes of T can be partitioned in 5 subsets, depending on their position relatively to n. Hence, for each node variable n appearing in a tree description, it is first associated with an integer (called node id), we then define the 5 sets of node ids (*i.e.*, sets of integers)

Down$_n$, *Up*$_n$, *Left*$_n$, *Right*$_n$ and *Eq*$_n$ referring to the ids of the nodes located below, above, on the left, on the right or identified with n respectively (see Figure 6). Note that we require that these sets are a partition of all node ids.

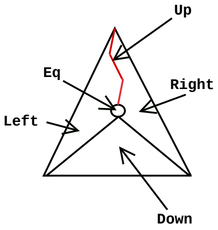


Figure 6
Partition of the nodes of tree models

Using this set-based representation of a model, we translate each node relation from the input formula (built on the tree description language introduced in Section 4) into constraints on the sets of node ids that must hold in a valid model. For instance, the sub-formula $n_1 \prec^+ n_2$ which states that node n_1 strictly precedes node n_2 is translated into:

$$n_1 \prec^+ n_2 \ \equiv \mathit{EqDown}_{n_1} \subseteq \mathit{Left}_{n_2} \wedge \mathit{EqDown}_{n_2} \subseteq \mathit{Right}_{n_1} \wedge \tag{24}$$

$$\mathit{Right}_{n_2} \subseteq \mathit{Right}_{n_1} \wedge \mathit{Left}_{n_1} \subseteq \mathit{Left}_{n_2} \tag{25}$$

where[15] $\mathit{EqDown}_x = \mathit{Eq}_x \uplus \mathit{Down}_x$ for $x \in \{n_1, n_2\}$. In other words, in a valid minimal tree model, the set of nodes below or equal to n_1 is included in the set of nodes (strictly) on the left of n_2, the set of nodes below or equal to n_2 is included in the set of nodes (strictly) on the right of n_1, the set of nodes on the right of n_2 is included in the set of nodes on the right of n_1 and finally the set of nodes on the left of n_1 is included in the set of nodes on the left of n_2.

Once all input relations are translated into set constraints, the solver uses standard Constraint Satisfaction techniques (e.g., a *first-fail* exploration of the search tree) to find a set of consistent partitions. Finally, the nodes of the models are obtained by considering nodes with distinct Eq_n.

FB-LTAG trees. To support the specification of FB-LTAG trees, the XMG compiler extends the generic tree solver described above with a set of constraints ensuring that the trees are well-formed TAG trees. In effect, these constraints require the trees to be linear ordered trees with appropriate decorations. Each node must be labelled with a syntactic category. Leaf nodes are either terminal, foot or substitution nodes. There is at most one foot node per tree and the category of the foot node must be identical to that of the root node. Finally, each tree must have at least one leaf node which is an anchor.

MCTAG tree sets. While FB-LTAG consists of trees, Multi-Component Tree-Adjoining Grammar (MC-TAG) (Weir 1988) consists of sets of trees. To support the specification of MC-TAG, the sole extension needed concerns node variables that are not dominated by any other node variable in the tree description. Whilst for FB-LTAG, these are taken

[15] $\uplus$ represents disjoint union.

to denote either the same root node or nodes that are connected to some other node (*i.e.*, uniqueness of the root), for MC-TAG they can be treated as distinct nodes thereby allowing for models that are sets of trees rather than trees (Parmentier et al. 2007). In other words, the only modification brought to the tree description solver is that, in MC-TAG mode, it does not enforce the uniqueness of a root node in a model.

IG polarized tree descriptions. Interaction Grammars (Perrier 2000) consist of tree descriptions whose node variables are labelled with polarized feature structures. A polarized feature structure is a set of polarized feature triples (f, p, v) where f and v are standard features and feature values respectively while p is a polarity value in $\{\rightarrow, \leftarrow, =, \approx\}$. Polarities are used to guide parsing in that a valid derivation structure must neutralize polarities.

To support an XMG encoding of IG, two extensions are introduced namely (i) the ability to output tree descriptions rather than trees, and (ii) the ability to write polarized feature structures. The first extension is trivially realized by specifying a description solver which ensures that any output description has at least one tree model. For the second point, the *SYN* language is extended to define polarized feature structures and the unification engine to support unification of polarized features (for instance, a $\rightarrow$ feature will unify with a neutral $(=)$ feature to yield a $\rightarrow$ polarized feature value triple).

5.3.2 Adding specific linguistic constraints: the case of clitics. XMG can be extended to support specific constraints on tree descriptions (*e.g.*, constraints on node linear order), which make it possible to describe linguistic-dependent phenomena, such as for instance clitic ordering in French, at a meta-level (*i.e.*, within the metagrammar).

According to Perlmutter (1970), clitics are subject to two hard constraints. First, they appear in front of the verb in a fixed order according to their rank[16] (26a-26b). Second, two different clitics in front of the verb cannot have the same rank (26c).

(26) a. Jean le$_3$ lui$_4$ donne.
'John gives it to him.'
b. *Jean lui$_4$ le$_3$ donne.
*'John gives to him it.'
c. *Jean le$_3$ la$_3$ donne.
*'John gives it it.'

To support a direct encoding of Perlmutter's observation, XMG includes both a *node uniqueness principle* and a *node ordering principle*. The latter allows us to label nodes with some property (let us call it *rank*) whose value is an integer (for instance, one can define a node as $n_1(rank : 2)[cat : Cl]$). When solving tree descriptions, XMG further requires that in a valid tree model, (i) there are no two nodes with the same rank and (ii) sibling nodes labelled with a rank are linearly ordered according to their rank.

Accordingly, in the French grammar of Crabbé (2005), each node labelled with a clitic category is also labelled with a numerical node property representing its rank.[17] XMG ordering principle then ensures that the ill-formed tree crossed out in the Figure 7, is not produced. Note that in Figure 7, every type of clitic is defined locally (*i.e.*, in a

[16] The number on the clitics indicate their rank.

[17] Recall that node properties are features whose values are used by the tree description solver in order to restrict the set of valid models. These properties may not appear in the trees produced from the input metagrammar. For instance, the rank property is not part of the FB-LTAG formalism, and thus does not appear in the FB-LTAG elementary trees produced by XMG.

separate class), and that the interactions between these local definitions are handled by XMG using this rank principle, to produce only one valid description (pictured on the right of the arrow).

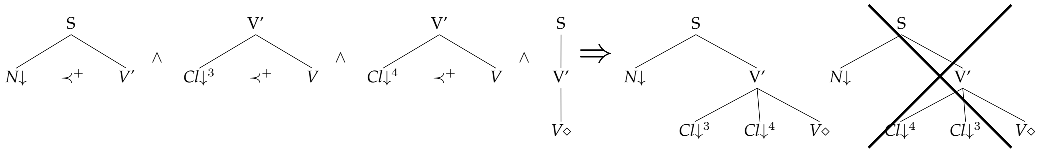


Figure 7
Clitic ordering in French

That is, XMG ordering constraints permit a simple, declarative, encoding of the interaction between clitics. This again contrasts with systems based on lexical rules. As noted by Perlmutter (1970), if clitics are assumed to be moved by transformations, then the order in which lexical rules apply this movement must be specified.

To implement the uniqueness principle, one needs to express the fact that in a valid model ϕ, there is only one node having a given property p (*i.e.*, a parameter of the constraint, here the value of the *rank node property*). This can be done by introducing, for each node n of the description, a Boolean variable p_n indicating whether the node denoting n in the model has this property or not (*i.e.*, are there two nodes of identical rank ?). Then, if we call $\mathcal{V}_p^\phi$ the set of integers referring to nodes having the property p in a model, we have: $p_n \equiv (Eq_n \cap \mathcal{V}_p^\phi) \neq \emptyset$. Finally, if we represent p_n being *true* with 1 and p_n being *false* with 0,[18] and we sum the p_n for each n in the model, we have that in a valid model, this sum is strictly lower than 2: $\sum_{n \in \phi} p_n \; < \; 2$.

To implement the ordering principle, one needs to express the fact that in a valid model ϕ, two sibling nodes n_1 and n_2 having a given property p of type integer and of values p_1 and p_2 respectively, are such that the linear precedence between these nodes conform to the natural order between p_1 and p_2. This can be done, by first introducing, for each pair of nodes n, m of the description, a Boolean variable $b_{n,m}$ indicating whether they have the same ancestors: $b_{n,m} \equiv (Up_n \cap Up_m) = (Up_n \cup Up_m)$. For each pair of nodes that do so, we check whether they both have the property p, and if this is case, we add to the input description, a strict precedence constraint on these nodes according to their respective values of the p property:[19]

$$b_{n,m} \wedge (p_n < p_m) \; \Rightarrow \; n \prec^+ m \tag{27}$$

$$b_{n,m} \wedge (p_m < p_n) \; \Rightarrow \; m \prec^+ n \tag{28}$$

5.3.3 Adding color constraints to facilitate grammar writing. To further ease grammar development, XMG supports a node coloring mechanism that permits nameless node identification (Crabbé and Duchier 2004), reminiscent of the *polarity*- based node identification first proposed in Muskens and Krahmer (1998) and later used by Duchier and Thater (1999), Perrier (2000). Such a mechanism offers an alternative to explicit node identification using equations between node variables. The idea is to label node

[18] These integer representations are usually called *reified* constraints.

[19] In fact, rather than adding strict precedence constraints to the tree description, we directly add to the solver their equivalent set constraints on *Eq*, *Up*, *Left*, *Right*, *Down*, introduced above.

variables with a color property, whose value (either red, black or white) can trigger node identifications.

This mechanism is another parameter of the tree solver. When in use, the valid tree models must satisfy some color constraints, namely they must only have red or black nodes (no remaining white nodes, these have to be identified with some black node). As shown in the table below, node identification must observe the following constraints: a white node must be identified with a black node; a red node cannot be identified with any other node; and a black node may be identified with one or more white nodes.[20]

	$\bullet_B$	$\bullet_R$	$\circ_W$	$\perp$
$\bullet_B$	$\perp$	$\perp$	$\bullet_B$	$\perp$
$\bullet_R$	$\perp$	$\perp$	$\perp$	$\perp$
$\circ_W$	$\bullet_B$	$\perp$	$\circ_W$	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

We now briefly describe how the constraint solver sketched in Section 5.3.1 was extended to support colors. As mentioned above, in valid models, all the white nodes are identified with a black node. Consequently, there is a bijection from the red and black nodes of the tree description to the nodes of the model. In order to take this bijection into account, we add a node variable RB_n to the 5 sets already associated with a node variable n from section 5.1. RB_n denotes either n if n is a black or red node, or the black node identified with n if n is a white node. Note that all the node variables must be colored: the set of node variables in a tree description can then be partitioned into 3 sets Red, $Black$ and $White$. Basically, we know that, for all node n, $RB_n \in Eq_n$ (this is what the bijection is about). Again we translate color information into constraints on node sets (these constraints help the generic tree solver by reducing the ambiguity for the Eq_n sets):

$$n \in Red \Rightarrow (n = RB_n) \ \wedge \ (Eq_n = \{n\}) \tag{29}$$

$$n \in Black \Rightarrow (n = RB_n) \ \wedge \ (Eq_n \backslash \{n\} \subseteq White) \tag{30}$$

$$n \in White \Rightarrow (RB_n \in Black) \ \wedge \ (Eq_n \cap Black = \{RB_n\}) \tag{31}$$

Node coloring offers an alternative to complex namespace management. The main advantage of this particular identification mechanism is its economy: not only is there no longer any need to remember node identifiers, there is in fact no need to choose a name for node variables.

It is worth stressing that the XMG node identification process is reduced to a constraint solving problem and so it is not a sequential process. Thus the criticisms levelled by Cohen-Sygal and Wintner (2007, 2009) against non associative constraints on node unification do not apply.

Briefly, in their work, Cohen-Sygal and Wintner (2007, 2009) showed that any polarity-based tree description formalism is not associative. In other words, when describing trees in terms of combinations of polarized structures, the order in which the structures are combined matters (*i.e.*, the output structures depend on the combination order). This feature makes such formalisms not appropriate for a modular and collaborative grammar engineering, such as that of Cohen-Sygal and Wintner (2011) for Unification Grammar.

[20] In other words, node colors can be seen as information on node saturation.

In the XMG case, when using node colors, the tree description solver does not rely on any specific fragment combination order. It computes *all* possible combination orders. In this context, the grammar designer cannot think in terms of sequences of node identifications. This would lead to tree overgeneration.

Again, it is important to remember that tree solving computes any valid tree model, independently of any specific sequence of node identifications (all valid node identifications are computed). In this context, non associativity of color-based node identification is not an issue, but rather a feature, as it allows for a compact description of a large number of node identifications (and thus of tree structures).

6. Writing Grammars with XMG

In this section, we first provide a detailed example showing how XMG can be used to specify the verbal trees of a large FB-LTAG for French extended with a unification based semantics. We then give a brief description of several large and middle scale grammars that were implemented using XMG.

6.1 SEMTAG: a large FB-LTAG for French covering syntax and semantics

We now outline the XMG specification for the verbal trees of SEMTAG, a large FB-LTAG for French. This specification further illustrates how the various features of XMG (*e.g.*, combined use of disjunction and conjunction, node colors) permit us to specify compact and declarative grammar descriptions. We first discuss the syntactic dimension (*SYN*). We then go on to show how the semantic dimension (*SEM*) and the syntax/semantic interface (*DYN*) are specified.

6.1.1 The syntactic dimension. The methodology used to implement the verbal fragment of SEMTAG can be summarized as follows. First, tree fragments are defined which represent either a possible realization of a verb argument or a possible realization of the verb. The verbal elementary TAG trees of SEMTAG are then defined by appropriately combining these tree fragments.

To maximize structure sharing, we work with four levels of abstraction. First, basic tree fragments describing verb or verb argument realizations are defined. Second, grammatical functions are defined as disjunctions of argument realizations. Third, verbal diathesis alternatives are defined as conjunctions of verb realizations and grammatical functions. Fourth, diathesis are gathered into tree families. In the next paragraphs, we explain each of these levels in more details.

Tree fragments. Tree fragments are the basic building blocks used to define SEMTAG. These are the units that are shared and reused in the definition of many elementary trees. For instance, the fragment for a canonical subject will be used by all FB-LTAG elementary trees involving a canonical subject.

As mentioned above, to specify the verbal elementary trees of SEMTAG, we begin by defining tree fragments which describe the possible syntactic realizations of the verb arguments and of the verb itself. Figure 8 provides some illustrative examples of these fragments. Here and in the sequel, we omit the feature structures decorating the trees to facilitate reading.[21]

[21] See (Crabbé 2005) for a complete description of SEMTAG tree fragments, including feature structures.

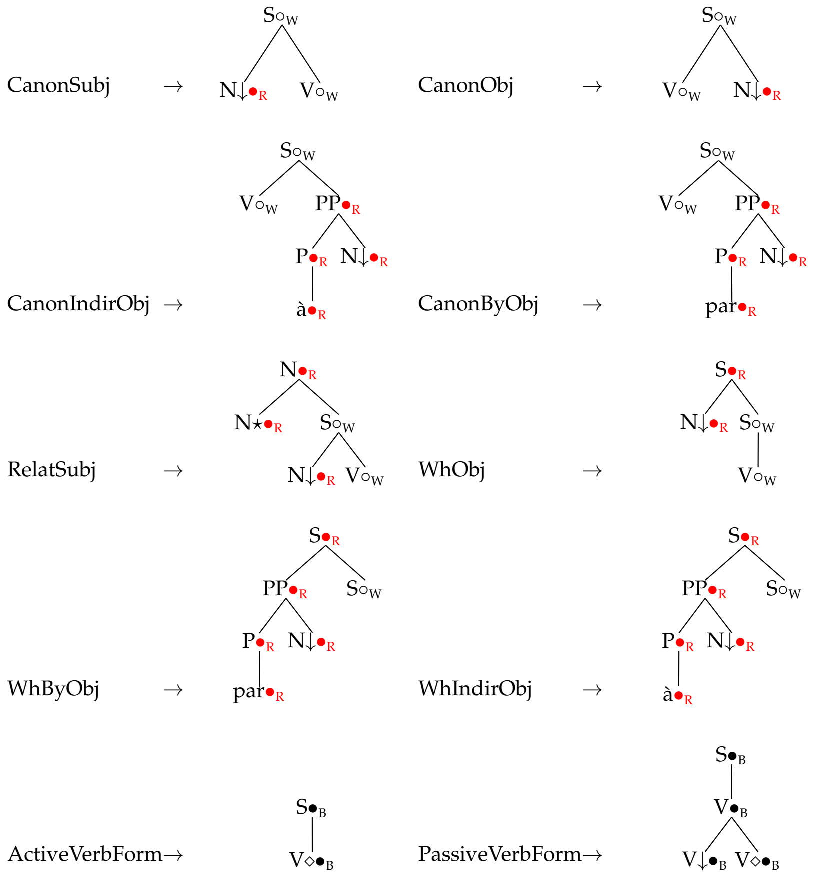


Figure 8
Elementary tree fragments used as building blocks of the grammar (nodes are colored to control their identification when blocks are combined)

To further factorize information and facilitate grammar maintenance, the basic tree fragments are organized in an inheritance hierarchy.[22] Figure 9 shows a partial view of this hierarchy illustrating how the tree fragments for argument realization depicted in Figure 8 are organized to maximize the sharing of common information. The hierarchy classifies the verbal arguments depicted in Figure 8 into four categories:

1. The canonical subject is a noun realized in front of the verb.
2. Canonical complements occur after the verb. The canonical object is a noun phrase whereas prepositional complements are introduced by specific prepositions namely, *à* for the canonical indirect object and *par* for the canonical by object.
3. Wh-arguments (or questioned arguments) occur in front of a sentence headed by a verb. A Wh-object is an extracted noun whilst questioned prepositional objects are extracted prepositional phrases that are introduced by a specific preposition.

[22] Recall from Section 4 that inheritance is used to share namespaces. Thus, (node or feature) variables introduced in a given class C can be directly reused in the sub-classes of C.

4. Finally, the relativized subject is a relative pronoun realized in front of the sentence. Extracted subjects in French cannot be realized at an unbounded distance of the predicate.

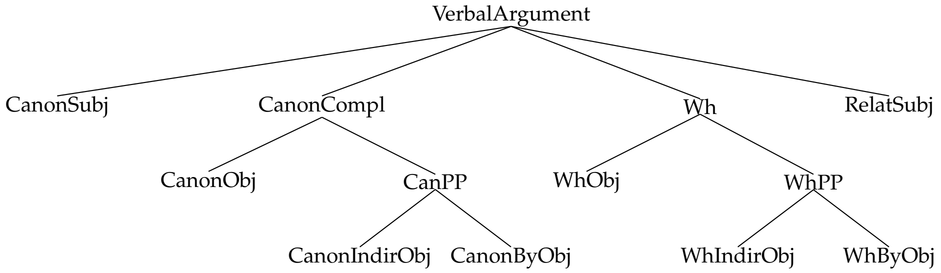


Figure 9
Organization of elementary fragments in an inheritance hierarchy

Syntactic functions. The second level of abstraction uses syntactic function names such as Subject and Object to group together alternative ways in which a given syntactic function can be realized. For instance, if we make the simplifying assumption that the possible argument realizations are limited to those given in Figure 8, the *Subject, Object, ByObject* and *IndirectObject* classes would be defined as follows.[23]

$$Subject \rightarrow CanonSubj \vee RelatSubj \tag{32}$$

$$Object \rightarrow CanonObj \vee WhObj \tag{33}$$

$$ByObject \rightarrow CanonByObj \vee WhByObj \tag{34}$$

$$IndirectObject \rightarrow CanonIndirObj \vee WhIndirObj \tag{35}$$

That is, we define the Subject class as an abstraction for talking about the set of tree fragments that represent the possible realizations of a subject argument namely, in our restricted example, canonical and relativized subject. Thus, the simplified *Subject* class defined above characterizes contexts such as the following :

(36) a. **Jean** mange. (canonical subject)
’**John** eats.’
b. Le garçon **qui** mange (relativized subject)
’The boy **who** eats’

Similarly, the *IndirectObject* class abstracts over the realization of an argument introduced by the preposition *à* to the right of the verb (CanonIndirObj) or realized in extracted position (possibly realized at an unbounded distance from the predicate) as illustrated by the following examples :

[23] Note that, when these abstractions will be combined to describe for instance transitive verbs, the combination of WhObj with WhByObj will be ruled out by using a uniqueness principle such as introduced in Section 5.

(37) a. Jean parle **à Marie**. (canonical indirect object)
'John talks **to Mary**.'
b. **À qui** Jean parle-t-il ? (wh indirect object)
'**To whom** is John talking ?'
c. **À qui** Pierre croit-il que Jean parle ? (wh indirect object)
'**To whom** Peter thinks that John talks ?'

This way of grouping tree fragments is reminiscent of the informal classification of French syntactic functions presented by Iordanskaja and Mel'čuk (2009) whereby each syntactic function is associated with a set of possible syntactic constructions.

Diathesis alternations. In this third level, we take advantage of the abstractions defined in the previous level to represent diathesis alternations. Again we are interested here in describing alternatives. Diathesis alternations are those alternations of mapping between arguments and syntactic functions such as for instance the active/passive alternation. In a diathesis alternation, the actual form of the verb constrains the way predicate arguments are realized in syntax. Thus, in the following example, it is considered that both (38a) and (38b) are alternative realizations of a predicate argument structure such as *send(John, a letter)*.

(38) a. Jean **envoie** une lettre.
'John sends a letter.'
b. Une lettre **est envoyée** par Jean.
'A letter is sent by John.'

The active/passive diathesis alternation captures the fact that if the verb is in the active form, its two arguments are realized by a subject and an object whereas if the verb is in the passive form, then the arguments consist of a subject and a by-object.

$$\begin{aligned} TransitiveDiathesis \quad &\rightarrow \quad (Subject \wedge ActiveVerbForm \wedge Object) \\ &\vee (Subject \wedge PassiveVerbForm \wedge ByObject) \end{aligned} \tag{39}$$

Finally a traditional case of "erasing"[24], such as the agentless passive (or passive without agent) can be expressed in our language by adding an additional alternative where the By Object or agentive complement is not expressed. Thus (40) is an augmentation of (39) where we have added the agentless passive alternative (indicated in bold face).

$$\begin{aligned} TransitiveDiathesis \quad &\rightarrow \quad (Subject \wedge ActiveVerbForm \wedge Object) \\ &\vee (Subject \wedge PassiveVerbForm \wedge ByObject) \\ &\vee (\mathbf{Subject} \wedge \mathbf{PassiveVerbForm}) \end{aligned} \tag{40}$$

This methodology can be further augmented to implement an actual linking in the line of Bresnan and Zaenen (1990). For the so-called erasing cases, one can map the "erased" predicative argument to an empty realization in syntax. We refer the reader to (Crabbé 2005) for further details.

[24] It is often argued that a language of grammatical representation must be equipped with an "erasing device" like lexical rules because of phenomena such as the passive without agent. In this framework it turns out that this kind of device is not needed since we do not grant any special status to base trees.

Tree families. Finally, tree families are defined that is, sets of trees capturing alternative realizations of a given verb type (*i.e.*, sub-categorization frame). Continuing with the simplified example presented so far, we can for instance define the tree family for verbs taking a nominal subject, a nominal object and an indirect nominal object (*i.e.*, ditransitive verbs) as follows :

$$DitransitiveFamily \quad \rightarrow \quad TransitiveDiathesis \wedge Indirectobject \tag{41}$$

The trees generated for such a family will, among others, handle the following contexts:[25]

(42) a. Jean offre des fleurs à Marie.
’John offers flowers to Mary.’
b. À quelle fille Jean offre-t-il des fleurs ?
’To which girl does John offer flowers ?’
c. Le garçon qui offre des fleurs à Marie.
’The boy who offers flowers to Mary.’
d. Quelles fleurs le garçon offre-t-il à Marie ?
’Which flowers does the boy offer to Mary ?’
e. Les fleurs sont offertes par Jean à Marie.
’The flowers are offered by John to Mary.’
f. Par quel garçon les fleurs sont-elles offertes à Marie ?
’By which boy the flowers are offered to Mary.’

It is straightforward to extend the grammar with new families. Thus, for instance (43) shows how to define the transitive family (for verbs taking a nominal subject and a nominal object) and (44), the intransitive one (alternatives of a verb sub-categorizing for a nominal subject).

$$TransitiveFamily \quad \rightarrow \quad TransitiveDiathesis \tag{43}$$

$$IntransitiveFamily \quad \rightarrow \quad Subject \wedge ActiveVerbForm \tag{44}$$

Similarly, tree families for non verbal predicates (adjectives, nouns) can be defined using the abstraction over grammatical functions defined for verbs. For instance, the examples in (45a-45b) can be captured using the adjectival trees defined in (47) and (48) respectively where *Subject* extends the definition of subject given above with a Wh-subject, *PredAdj* combines a subject tree fragment with a tree fragment describing a predicative adjective and *PredAdjAObj* extends a *PredAdj* tree fragment with a canonical à-object.

(45) a. Jean est attentif, Qui est attentif?, L’homme qui est attentif.
’John is mindful, Who is mindful?, The man who is mindful.’
b. Jean est attentif à Marie, Qui est attentif à Marie?, L’homme qui est attentif à Marie.
’John is mindful of Mary, Who is mindful of Mary?, The man who is mindful of Mary.’

[25] Note that number and gender agreements are dealt with using coreferences between features labelling syntactic nodes, see (Crabbé 2005).

$$Subject \rightarrow CanonSubj \vee RelatSubj \vee WhSubj \tag{46}$$

$$PredAdj \rightarrow Subject \wedge AdjectivalForm \tag{47}$$

$$PredAdjAObj \rightarrow PredAdj \wedge CanonAObj \tag{48}$$

6.1.2 The semantic dimension and the Syntax / Semantic Interface. We now show how to extend the XMG specification presented in the previous section to integrate a unification-based compositional semantics. Three main changes need to be carried out:

1. Each elementary tree must be associated with a semantic formula. This is done using the *SEM* dimension.
2. The nodes of elementary trees must be labelled with the appropriate semantic indices. This involves introducing the correct attribute-value pair in the correct feature structure (top or bottom) on the appropriate node.
3. Syntax and semantics need to be synchronized that is, variable sharing between semantic formulae and tree indices need to be enforced. To this end we use the *DYN* dimension.

Informing the Semantic Dimension. To associate each elementary tree with a formula representing the meaning of the words potentially anchoring that tree, we use the *SEM* dimension to specify a semantic schema. For instance, the *TransitiveFamily* class defined in (43) for verbs taking two nominal arguments is extended as follows :

$$TransitiveFamily \rightarrow TransitiveDiathesis \wedge BinaryRel \tag{49}$$

where *TransitiveDiathesis* is the XMG class defined in (40) to describe the set of trees associated with transitive verbs and *BinaryRel* the class describing the following semantic schema :

$$L : P(E) \wedge L : Theta_1(E, X) \wedge L : Theta_2(E, Y) \tag{50}$$

In this semantic schema, P, $Theta_1$ and $Theta_2$ are unification variables that become ground when the tree is anchored with a specific word. For instance, P, $Theta_1$ and $Theta_2$ are instantiated to *eat*, *agent* and *patient* respectively when the anchor is *ate* (these pieces of information – predicate, thematic roles – are associated with lemmas, located in the syntactic lexicon, and unified with adequate semantic variables via anchoring equations). Further, X, Y, E, L are unification variables representing semantic arguments. As illustrated in Figure 3, these become ground during (or after) derivation as a side effect of the substitutions and adjunctions taking place when trees are combined. It is worth noting that by combining semantic schemas with diathesis classes, one such specification assigns the specified semantic schema to many trees namely, all the trees described by the corresponding diathesis class. In this way, the assignment of semantic formulae to trees is relatively economical. Indeed in SemTag, roughly 6,000 trees are assigned a semantic schema using a total of 75 schema calls.

Co-indexing Trees and Formulae Indices. Assuming that tree nodes are appropriately decorated with semantic indices by the specification scheme described in the next paragraph, we now show how to enforce the correct mapping between syntactic and semantic arguments. This is done in two steps.

First, we define a set of interface constraints of the form $\langle index_F : V, arg_i : V\rangle$ which are used to enforce the identification of the semantic index ($index_F$) labelling a given tree node with grammatical function F (*e.g.*, F := subject) with the index (arg_i) representing the i-th argument in a semantic schema. For instance, the following constraints ensure a *subject/arg*$_1$ mapping that is, a coreference between the index labelling a subject node and the index representing the first argument of a semantic schema:

$$\begin{array}{rcl} C_1 & \rightarrow & Node\,[idx : I] \wedge \langle index_{subject} : I\rangle \\ C_2 & \rightarrow & L : P(E) \wedge L : Theta_1(E, X) \wedge \langle arg_1 : X\rangle \\ SubjectArg_1 & \rightarrow & C_1 \quad \wedge \quad C_2 \quad \wedge \langle index_{subject} : V, arg_1 : V\rangle \end{array} \tag{51}$$

Given such interface constraints, we refine the diathesis definitions so as to ensure the correct bindings. For instance, the specification in (39) is modified to:

$$\begin{array}{rcl} TransitiveActive & \rightarrow & (SubjectArg_1 \ \wedge \ ObjectArg_2 \wedge \\ & & \quad Subject \wedge ActiveVerbForm \wedge Object) \end{array} \tag{52}$$

whilst the passive diathesis is specified as:

$$\begin{array}{rcl} TransitivePassive & \rightarrow & (SubjectArg_2 \ \wedge \ ByObjectArg_1 \wedge \\ & & \quad Subject \wedge PassiveVerbForm \wedge ByObject) \end{array} \tag{53}$$

Labelling Tree Nodes with Semantic Indices. The above scheme relies on the assumption that tree nodes are appropriately labelled with semantic indices (*e.g.*, the subject node must be labelled with a semantic index) and that these indices are appropriately named (arg_1 must denote the parameter representing the first argument of a binary relation and $index_{subject}$ the value of the index feature on a subject node). As suggested by Gardent (2007), a complete semantic labelling of a TAG with the semantic features necessary to enrich this TAG with the unification based compositional semantics sketched above can be obtained by applying the following *labelling principles*[26]:

Argument labelling: In trees associated with semantic functors, each argument node is labelled with a semantic index[27] named after the grammatical function of the argument node (*e.g.*, $index_{subject}$ for a subject node).

Controller/Controllee: In trees associated with control verbs, the semantic index of the controller is identified with the value of the controlled index occurring on the sentential argument node.

Anchor projection: The anchor node projects its index up to its maximal projection.

[26] The principles required to handle quantification are omitted. We refer the reader to (Gardent 2007) for a more extensive presentation of how semantics is implemented using XMG.

[27] For simplicity, we only mention indices. However, to be complete, labels should also be used.

Foot projection: A foot node projects its index up to the root.[28]

As we shall now see, XMG permits a fairly direct encoding of these principles.

The Argument Labelling principle states that, in the tree associated with a syntactic functor (*e.g.*, a verb), each node representing a syntactic argument (*e.g.*, the subject node) should be labelled with a semantic index named after the grammatical function of that node (*e.g.*, $index_{subject}$).[29]

To specify this labelling, we define for each grammatical function $Function \in \{Subject, Object, ByObject, IndirectObject, \dots\}$, a semantic class $FunctionSem$ which associates with an (exported) node variable called $FunctionNode$ the feature value pair *[index : I]* and a *DYN* constraint of the form $\langle index_{Function} : I\rangle$. For instance, the class $SubjectSem$ associates the node $SubjectNode$ with the feature value pair *[index : I]* and the *DYN* constraint $\langle index_{subject} : I\rangle$.

$$SubjectSem \quad \rightarrow \quad SubjectNode\,[index : I] \;\wedge\; \langle index_{subject} : I\rangle \tag{54}$$

Additionally, in the tree fragments describing the possible realizations of the grammatical functions, the (exported) variable denoting the argument node is systematically named $ArgNode$.

Finally, we modify the specification of the realizations of the grammatical functions to import the appropriate semantic class and identify $ArgNode$ and $FunctionNode$. For instance, the *Subject* specification given above is changed to :

$$\begin{aligned} Subject \quad \rightarrow \quad & SubjectSem \;\wedge\; ArgNode = SubjectNode \;\wedge \\ & (CanonSubj \vee RelatSubj \vee WhSubj) \end{aligned} \tag{55}$$

As a result, all $ArgNode$ nodes in the tree descriptions associated with a subject realization are labelled with an index feature I whose global name is $index_{subject}$.

Value sharing between the semantic index of the controller (*e.g.*, the subject of the control verb) and that of the controllee (*e.g.*, the empty subject of the infinitival complement) is enforced using linking constraints between the semantic index labelling the controller node and that labelling the sentential argument node of the control verb. Control verb definitions then import the appropriate (object or subject control) linking constraint.

The anchor (resp. foot) projection principle stipulates the projection of semantic indices from the anchor (resp. foot) node up to the maximal projection (resp. root). Concretely, this means that the top and bottom features of the nodes located on this path between the anchor (resp. foot) and the maximal projection (resp. root) all include an index feature whose value is shared between adjacent nodes (see variables E_i in Figure 10)[30]. Once the top and bottom structures are unified, so are the semantic indices along this path (modulo expected adjunctions realized on the projection).

[28] The foot projection principle only applies to foot nodes that are not argument nodes (*i.e.*, to modifiee nodes).

[29] In other words, this argument labelling principle defines an explicit and normalized reference to any realization of a semantic argument. Following FB-LTAG predicate-argument co-occurrence principle (Abeillé, Candito, and Kinyon 1999), we know that any elementary tree includes a leaf node for each realized semantic argument of its anchor. This principle thus holds in any FB-LTAG. However, its implementation is closely related to the architecture of the metagrammar, here we benefit from the fact that verbal arguments are described in dedicated classes to reach a high degree of factorization.

[30] For sake of brevity, we write $\frac{E_2}{E_1}$ for $[bot : [index : E_1]\; top : [index : E_2]]$. $\langle\, \rangle$ refers to the anchor / foot.

To implement these principles, we define a set of anchor projection classes $\{Depth1, Depth2, Depth3\}$ as illustrated in Figure 10. We then "glue" these projection skeletons onto the relevant syntactic trees by importing the skeletons in the syntactic tree description and explicitly identifying the anchor node of the semantic projection classes with the anchor or foot node of these syntactic tree descriptions. Since the models must be trees, the nodes dominating the anchor node of the projection class will deterministically be identified with those dominating the anchor or foot node of the trees being combined with. For instance, for verbs, the class specifying the verbal spine (*e.g.*, *ActiveVerbForm*, see Figure 10) equates the anchor node of the verbal spine with that of the projection skeleton. As a result, the verb projects its index up to the root.

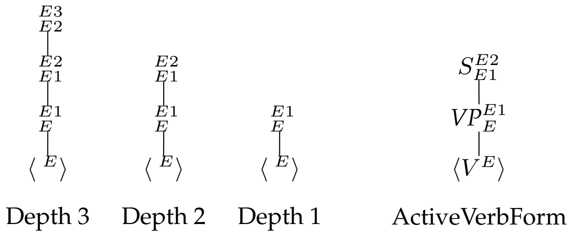


Figure 10
Anchor/Foot projection.

6.1.3 Some figures about SemTag**.** As mentioned above, SemTag is a large FB-LTAG for French equipped with semantics (Gardent 2008), it extends the purely syntactic Ftag of Crabbé (2005) with a unification based compositional semantics as described by Gardent and Kallmeyer (2003).[31] The syntactic Ftag in essence implements Abeillé (2002)'s proposal for an FB-LTAG-based modelling of French syntax. Ftag contains around 6,000 elementary trees built from 293 XMG classes and covers some 40 basic verbal sub-categorization frames. For each of these frames, Ftag defines a set of argument alternations (active, passive, middle, neuter, reflexivization, impersonal, passive impersonal) and of argument realizations (cliticization, extraction, omission, permutations, etc.) possible for this frame. Predicative (adjectival, nominal and prepositional) and light verb constructions are also covered as well as some common sub-categorizing noun and adjective constructions. Basic descriptions are provided for the remaining constructions namely, adverbs, determiners and prepositions.

Ftag and SemTag were both evaluated on the *Test Suite for Natural Language Processing* (Tsnlp) (Lehmann et al. 1996), using a lexicon designed specifically on the test suite, hence reducing lexical ambiguity (Crabbé 2005; Parmentier 2007). This test suite focuses on difficult syntactical phenomena, providing grammatical and ungrammatical sentences. These competence grammars accept 76 % of the grammatical items, reject 83% of the ungrammatical items and have an average ambiguity of 1.64 parses per sentence. To give an idea of the compilation time, under architectures made of a 2 Ghz processor with 1 Gb of RAM, it takes XMG 10 minutes to compile the whole

[31] Ftag and SemTag are freely available under the terms of the GPL-compliant CeCILL license, the former at `https://sourcesup.renater.fr/scm/viewvc.php/trunk/METAGRAMMARS/FrenchTAG/?root=xmg`, and the latter on request.

SemTag (recall that there is no semantic description solving, hence the compilation times between Ftag and SemTag do not differ).[32]

Note that SemTag can be used for assigning semantic representations to sentences when combined with an FB-LTAG parser and a semantic construction module as described by Gardent and Parmentier (2005, 2007).[33] Conversely, it can be used to verbalize the meaning denoted by a given semantic representation when coupled with the GenI surface realizer described by Gardent and Kow (2007).

6.2 Other grammars designed with XMG

XMG has been used mainly to design FB-LTAG and IG for French or English. More recently, it has also been used to design a FB-LTAG for Vietnamese and a TT-MCTAG for German. We now briefly describe each of these resources.

SemXTAG. The English grammar, SemXTag (Alahverdzhieva 2008), reimplements the FB-LTAG developed for English at the University of Pennsylvania (XTAG Research Group 2001) and extends it with a unification based semantics. It contains 1,017 trees and covers the syntactic fragment of XTAG namely auxiliaries, copula, raising and small clause constructions, topicalization, relative clauses, infinitives, gerunds, passives, adjuncts, ditransitives (and datives), ergatives, it-clefts, wh-clefts, PRO constructions, noun-noun modification, extraposition, determiner sequences, genitives, negation, noun-verb contractions, sentential adjuncts, imperatives and resultatives. The grammar was tested on a handbuilt test-suite of 998 sentences illustrating the various syntactic constructions meant to be covered by the grammar. All sentences in the test-suite can be parsed using the grammar.

FrenchIG. The extended XMG framework was used to design a core IG for French consisting of 2,059 tree descriptions compiled out of 448 classes (Perrier 2007). The resulting grammar is lexicalised, and its coverage was evaluated using the previously mentioned Tsnlp. The French IG accepts 88% of the grammatical sentences and rejects 85% of the ungrammatical sentences, although the current version of the French IG does not yet cover all the syntactic phenomena presented in the test suite (for example, causative and superlative constructions).

Vietnamese TAG. The XMG language was used by Le Hong, N'Guyen, and Roussanaly (2008) to produce a core FB-LTAG for Vietnamese. Their work is rather a proof of concept than a large-scale implementation. They focused on Vietnamese's categorization frames, and were able to produce a TAG covering the following frames: intransitive (tree family N0V), transitive with a nominal complement (N0VN1), transitive with a clausal complement (N0VS1), transitive with modal complement (N0V0V1), ditransitive (N0VN1N2), ditransitive with a preposition (N0VN1ON2), ditransitive with a verbal complement (N0V0N1V1), ditransitive with an adjectival complement (N0VN1A), movement verbs with a nominal complement (N0V0V1N1), movement verbs with an adjectival complement (N0V0AV1), and movement ditransitive (N0V0N1V1N2).

[32] As a comparison, about one hour was needed by Candito's compiler, to produce a French FB-LTAG containing about 1,000 tree schemas.

[33] As an alternative way to parse FB-LTAG grammars equipped with flat semantics such as those produced by XMG, one can use the Tübingen Linguistic Parsing Architecture (TuLiPA) (Kallmeyer et al. 2010).

GerTT. Another XMG-based grammar corresponds to the German MC-TAG of Kallmeyer et al. (2008). This grammar, called *GerTT*, is in fact an MC-TAG with Tree Tuples (Lichte 2007). This variant of MCTAG has been designed to model free word order phenomena. This is done by imposing node sharing constraints on MCTAG derivations (Kallmeyer 2005). GerTT covers phenomena such as scrambling, coherent constructions, relative clauses, embedded questions, copula verbs, complementized sentences, verbs with various sub-categorization frames, nouns, prepositions, determiners, adjectives, and partly includes semantics. It is made of 103 tree tuples, compiled from 109 classes.

7. Related Work

We now compare XMG with existing environments for designing tree-based grammars and briefly report on the grammars designed with these systems.

7.1 Environments for designing tree-based grammars

Candito's metagrammar compiler. The concept of metagrammar was introduced by Candito (1996). In her paper, Candito presented a compiler for abstract specifications of FB-LTAG trees (the so-called metagrammars). Such specifications are based on three dimensions, each of them being encoded in a separate inheritance hierarchy of linguistic descriptions. The first dimension describes *canonical sub-categorization frames* (*e.g.*, transitive), the second *redistributions of syntactic functions* (*e.g.*, active to passive), and the third the tree descriptions corresponding to the *realizations of the syntactic functions* defined in dimension 2. This three-dimensional metagrammatical description is then processed by a compiler to compute FB-LTAG tree schemas. In essence, these tree schemas are produced by associating a canonical sub-categorization frame (dimension 1) with a compatible redistribution schema (dimension 2), and with exactly one function realization (dimension 3) for each function required by the sub-categorization frame.

Candito's approach improves on previous proposals by Vijay-Shanker and Schabes (1992), Evans, Gazdar, and Weir (1995), in that it provides a linguistically principled basis for structuring the inheritance hierarchy. As shown in Section 6.1, the XMG definition of SEMTAG uses similar principles. Candito's approach differs however from the XMG account in several important ways:

- Much of the linguistic knowledge used to determine which classes to combine is hard-coded in the compiler (unlike in XMG, there is no explicit control on class combinations). In other words, there is no clear separation between the linguistic knowledge needed to specify a high-level FB-LTAG description and the algorithm used to compile an actual FB-LTAG from this description. This makes grammar extension and maintenance by linguists extremely difficult.
- As in (Vijay-Shanker and Schabes 1992; Evans, Gazdar, and Weir 1995), the linguistic description is non monotonic in that some erasing classes are used to remove information introduced by other dimensions (*e.g.*, agentless passive).
- The approach fails to provide an easy means to state exceptions. These are usually encoded in the compiling algorithm.
- The tree description language used to specify classes in dimension 3 relies on global node variables. Thus, two variables with identical names introduced in different classes are expected to refer to the same tree node. As argued in Section 4, this makes it hard to design large-scale metagrammars.

The LexOrg system. An approach similar to Candito's was presented by Xia et al. (1998), Xia (2001), Xia, Palmer, and Vijay-Shanker (2005, 2010). As in Candito's approach, a TAG abstract specification relies on a three-dimensional description made of namely, sub-categorization frames, blocks and lexical redistribution rules. To compile this specification into a TAG, the system selects a canonical sub-categorization frame, applies some lexical redistribution rules to derive new frames and finally select blocks corresponding to the resulting frames. These blocks contains tree descriptions using the logic of Rogers and Vijay-Shanker (1994).

LexOrg suffers from similar limitations as Candito's compiler. Much of the linguistic knowledge is embedded in the compiling algorithm making it difficult for linguists to extend the grammar description and to handle exceptions. Unlike in Candito's framework, the tree description language uses local node variables and let the tree description solver determine node identifications. Although this avoids having to memorize node names, this requires that the descriptions be constrained enough to impose the required node identifications and prevent the unwanted ones. In practice, this again complicates grammar writing. In contrast, XMG provides an intermediate solution which, by combining local variables with export declarations, avoids having to memorize too many node variable names (only those local to the relevant sub-hierarchy need memorizing) while permitting explicit node identification.

The metagrammar compiler of Gaiffe et al. Gaiffe, Crabbé and Roussanaly (2002) proposed a compiler for FB-LTAG that aims to remedy both the lack of a clear separation between linguistic information and compilation algorithm, and the lack of explicit control on the class combinations prevalent in (Candito 1996; Xia et al. 1998; Xia 2001). In their approach, the linguistic specification consists of a single inheritance hierarchy of classes, each class containing a tree description. The description logic used is similar to Candito's. That is, global node names are used. To trigger class combinations, classes are labelled with two types of information: *needs* and *resources*. The compiler selects all final classes of the hierarchy, performs all possible combinations and only keeps these combinations that neutralize the stated needs and resources. The tree descriptions contained in these neutral combinations are then solved to produce the expected trees.

While this approach implements a clear separation between linguistic information and compilation algorithm, the fully automatic derivation of FB-LTAG trees from the inheritance hierarchy makes it difficult in practice to control overgeneration. In contrast, XMG's explicit definitions of class combinations by conjunction, disjunction and inheritance makes it easier to control the tree set that will be generated by the compiler from the grammar specification. Additionally, the issues raised by global variables remain (no way to instantiate twice a given class, and cumbersome definition of variables in large metagrammar).

The MGCOMP system. More recently, Villemonte de la Clergerie (2005; 2010) proposed a compiler for FB-LTAG which aims at preserving a high degree of factorization in both the abstract grammar specification and the grammar which is compiled from it. Thus, the MGCOMP system does not compute FB-LTAG elementary trees, but factorized trees.

In MGCOMP, like in Gaiffe, Crabbé, and Roussanaly (2002)'s approach, a metagrammar consists of a single hierarchy of classes. The classes are labelled with needs and resources, and final classes of the hierarchy are combined to compute tree descriptions. The main differences with Gaiffe, Crabbé, and Roussanaly (2002), lies in the fact that (i) a description can include new factorizing operators, such as repetition (*Kleene-star operator*), shuffling (interleaving of nodes), optionality and disjunctions, and (ii) it offers namespaces to specify the scope of variables. MGCOMP's extended tree descriptions are not completely solved by the compiler. Rather, it compiles underspecified trees (also called factorized trees). With this approach, a large grammar is much smaller in terms of number of grammatical structures, than a classical FB-LTAG. As a result, the grammars it compiles are only compatible with the DyALog parsing environment (Villemonte de La Clergerie 2005). And, since the linguist designs factorized trees and not actual TAG trees, debugging the metagrammar becomes harder.

7.2 Resources built using Candito, Xia and De La Clergerie's systems

Candito's system has been used by Candito (1999) herself to design a core FB-LTAG for French and Italian, and later by Barrier (2006) to design a FB-LTAG for adjectives in French. Xia's system (LexOrg) has been used to semi-automatically generate XTAG (Xia 2001). De La Clergerie's system (MGCOMP) has been used to design a grammar for French named FRMG (FRench MetaGrammar) (Villemonte de la Clergerie 2010). FRMG makes use of MGCOMP's factorizing operators (*e.g.*, shuffling operator), thus producing not *sensu stricto* a FB-LTAG, but a factorized FB-LTAG. FRMG is freely available, contains 207 factorized trees (*e.g.*, having optional branches, etc.) built from 279 metagrammatical classes, and covers 95% of the TSNLP.

8. Conclusion

In this article, we presented the *eXtensible MetaGrammar* framework and argued that, contrary to other existing grammar writing environments for tree-based grammar, XMG is declarative, extensible and notationally expressive. We believe that these features make XMG particularly appropriate for a fast prototyping of the kind of deep tree-based grammars that are used in applications requiring high precision in grammar modelling (*e.g.*, language teaching, man/machine dialogue systems, data-to-text generation).

The XMG language is documented on-line, its compiler is open source software, freely available under the terms of the GPL-compliant CeCILL license.[34] Many grammars designed with XMG (FB-LTAG and IG for French and English, TT-MCTAG for German) are also open-source and available on-line.[35]

Future research will focus on extensibility. So far, XMG has been used to design tree-based grammars for different languages. We plan to extend XMG to handle other types of formalisms[36] such as dependency grammars, and to support dimensions other than syntax and semantics such as for instance, phonology or morphology. As mentioned in the article, XMG offers a modular architecture, making it possible to extend it relatively easily. Nonetheless, in its current state, such extensions imply modifying XMG's code. We are exploring new extensions of the formalism, which would allow the linguist to dynamically define her/his metagrammar formalism (*e.g.*, which principles or descriptions to use) depending on the target formalism.

Another interesting question concerns cross-language grammar engineering. So far, the metagrammar allows for dealing with structural redundancy. As pointed out by

[34] See `https://sourcesup.renater.fr/xmg`.

[35] The French TAG and French and English IG are available on XMG's website, the German TT-MCTAG is available at `http://www.sfs.uni-tuebingen.de/emmy/res.html`.

[36] Preliminary work on cross-framework grammar engineering has been realized by Clément and Kinyon (2003), who used Gaiffe *et al.*'s compiler to produce both a TAG and a LFG from a given metagrammar.

Kinyon et al. (2006), a metagrammar can be used to capture generalizations across languages and is surely worth further investigating.

Finally, we plan to extend XMG with features borrowed from Integrated Development Environments (IDE) for programming languages. Designing a grammar is, in some respect, similar to programming an application. Grammar environments should benefit from the same tools as those used for the development of applications (*e.g.*, incremental compilation, debugger, *etc*.).

Acknowledgments

We are grateful to the three anonymous reviewers for their valuable comments. Any remaining error is ours.

References

Abeillé, A. 2002. *Une grammaire électronique du français*. CNRS Editions.

Abeillé, A., M. Candito, and A. Kinyon. 1999. Ftag: current status and parsing scheme. In *Proceedings of Vextal '99*, pages 283–292, Venice, Italy.

Aït-Kaci, Hassan. 1991. *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press.

Alahverdzhieva, Katya. 2008. XTAG using XMG. Master Thesis, Nancy Université.

Baldridge, Jason, Sudipta Chatterjee, Alexis Palmer, and Ben Wing. 2007. DotCCG and VisCCG: Wiki and programming paradigms for improved grammar engineering with OpenCCG. In Tracy Holloway King and Emily M. Bender, editors, *Proceedings of the Grammar Engineering Across Framework Workshop (GEAF 07)*, pages 5–25, Stanford, CA. CSLI.

Barrier, Sébastien. 2006. *Une métagrammaire pour les noms prédicatifs du français : développement et expérimentations pour les grammaires TAG*. Ph.D. thesis, Université Paris 7.

Becker, Tilman. 1993. *HyTAG: A new Type of Tree Adjoining Grammars for Hybrid Syntactic Representation of Free Word Order Language*. Ph.D. thesis, Universität des Saarlandes.

Blackburn, Patrick, Johan Bos, and Kristina Striegnitz. 2006. *Learn Prolog Now!*, volume 7 of *Texts in Computing*. College Publications.

Bresnan, Joan and Annie Zaenen. 1990. Deep unaccusitivity in LFG. In K. Dziwirek, P. Farell, and E. Mejias-Bikandi, editors, *Grammatical Relations : A Cross-Theoretical Perspective*. CSLI publication, Stanford, pages 45–57.

Candito, Marie. 1996. A principle-based hierarchical representation of LTAGs. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, pages 194–199, Copenhagen, Denmark.

Candito, Marie. 1999. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées : application au français et à l'italien*. Ph.D. thesis, Université Paris 7.

Clément, Lionel and Alexandra Kinyon. 2003. Generating parallel multilingual lfg-tag grammars from a metagrammar. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 184–191, Sapporo, Japan, July.

Cohen-Sygal, Yael and Shuly Wintner. 2007. The Non-Associativity of Polarized Tree-Based Grammars. In *Proceedings of the Eighth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2007)*, pages 208–217, Mexico City, Mexico.

Cohen-Sygal, Yael and Shuly Wintner. 2009. Associative Grammar Combination Operators for Tree-Based Grammars. *Journal of Logic, Language and Information*, 18:293–316.

Cohen-Sygal, Yael and Shuly Wintner. 2011. Towards modular development of typed unification grammars. *Comput. Linguist.*, 37(1):29–74.

Copestake, Ann and Dan 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.

Copestake, Ann, Alex Lascarides, and Dan Flickinger. 2001. An algebra for semantic construction in constraint-based grammars. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 140–147, Toulouse, France, July. Association for Computational Linguistics.

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

Crabbé, Benoît and Denys Duchier. 2004. Metagrammar redux. In Henning Christiansen, Peter Rossen Skadhauge, and Jørgen Villadsen, editors, *proceedings of the Workshop on Constraint Solving for Language Processing (CSLP 2004)*, volume 3438 of *Lecture Notes in Computer Science, Springer*, pages 32–47, Copenhagen, Denmark.

Duchier, Denys, Brunelle Magnana Ekoukou, Yannick Parmentier, Simon Petitjean, and Emmanuel Schang. 2012. Describing Morphologically-rich Languages using Metagrammars: a Look at Verbs in Ikota. In *Workshop on "Language technology for normalisation of less-resourced languages", 8th SALTMIL Workshop on Minority Languages and 4th Workshop on African Language Technology, International Conference on Language Resources and Evaluation, LREC 2012*, Istanbul, Turkey, 05.

Duchier, Denys and Joachim Niehren. 2000. Dominance constraints with set operators. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 326–341. Springer.

Duchier, Denys, Yannick Parmentier, and Simon Petitjean. 2012. Metagrammars As Logic Programs. In *International Conference on Logical Aspects of Computational Linguistics (LACL 2012). Demo Session*, Nantes, France.

Duchier, Denys and Stefan Thater. 1999. Parsing with Tree Descriptions: a constraint-based approach. In *Proceedings of the Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP'99)*, pages 17–32, Las Cruces, New Mexico.

Evans, Roger, Gerald Gazdar, and David Weir. 1995. Encoding lexicalized tree adjoining grammars with a nonmonotonic inheritance hierarchy. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 77–84, Cambridge, Massachusetts, USA.

Flickinger, Daniel. 1987. *Lexical Rules in the Hierarchical Lexicon*. Ph.D. thesis, Stanford University.

Gaiffe, Bertrand, Benoît Crabbé, and Azim Roussanaly. 2002. A new metagrammar compiler. In *Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks (TAG+6)*, pages 101–108, Venice, Italy.

Gardent, Claire. 2007. Tree Adjoining Grammar, Semantic Calculi and Labelling Invariants. In *Proceedings of the International Workshop on Computational Semantics (IWCS)*, Tilburg, Netherlands.

Gardent, Claire. 2008. Integrating a unification-based semantics in a large scale Lexicalised Tree Adjoininig Grammar for French. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING'08)*, pages 249–256, Manchester, UK.

Gardent, Claire and Laura Kallmeyer. 2003. Semantic Construction in Feature-Based Tree Adjoining Grammar. In *Proceedings of the 10th conference of the European Chapter of the Association for Computational Linguistics*, pages 123–130, Budapest, Hungary.

Gardent, Claire and Eric Kow. 2007. A symbolic approach to Near-Deterministic Surface Realisation using Tree Adjoining Grammar. In *45th Annual Meeting of the Association for Computational Linguistics*, pages 328–335, Prague, Czech Republic. Association for Computational Linguistics.

Gardent, Claire and Yannick Parmentier. 2005. Large scale semantic construction for Tree Adjoining Grammars. In Philippe Blache, Edward P. Stabler, Joan Busquets, and Richard Moot, editors, *Proceedings of the Fifth International Conference on Logical Aspects of Computational Linguistics (LACL'05)*, volume 3492 of *Lecture Notes in Computer Science*, pages 131–146, Bordeaux, France. Springer.

Gardent, Claire and Yannick Parmentier. 2006. Coreference Handling in XMG. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL 2006) Main Conference Poster Sessions*, pages 247–254, Sydney, Australia. Association for Computational Linguistics.

Gardent, Claire and Yannick Parmentier. 2007. SemTAG: a platform for specifying Tree Adjoining Grammars and performing TAG-based Semantic Construction. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 13–16, Prague, Czech Republic.

Iordanskaja, Lidija and Igor Mel'čuk. 2009. Establishing an Inventory of Surface - Syntactic Relations: Valence - controlled Surface - dependents of the Verb in French. In: Polguère & Mel'čuk (eds). Observatoire de linguistique Sens-Texte, Université de Montréal.

Joshi, Aravind K., Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of the Computer and System Sciences*, 10:136–163.

Kallmeyer, Laura. 1999. *Tree Description Grammars and Underspecified Representations*. Ph.D. thesis, Universität Tübingen.

Kallmeyer, Laura. 2005. Tree-Local Multicomponent Tree-Adjoining Grammars with Shared Nodes . *Computational Linguistics*, 31(2):187–226.

Kallmeyer, Laura, Timm Lichte, Wolfgang Maier, Yannick Parmentier, and Johannes Dellert. 2008. Developing a TT-MCTAG for German with an RCG-based parser. In *Proceedings of the Sixth Language Resources and Evaluation Conference (LREC)*, pages 782–789, Marrakech, Morocco.

Kallmeyer, Laura, Wolfgang Maier, Yannick Parmentier, and Johannes Dellert. 2010. TuLiPA - Parsing Extensions of TAG with Range Concatenation Grammars. *Bulletin of the Polish Academy of Sciences : Technical Sciences*, 58(3):377–392.

Kallmeyer, Laura and Maribel Romero. 2004a. LTAG semantics for questions. In *In Proceedings of 7th Internationl Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+7)*, pages 186–193, Vancouver, Canada.

Kallmeyer, Laura and Maribel Romero. 2004b. LTAG semantics with semantic unification. In *Proceedings of 7th Internationl Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+7)*, page 155–162, Vancouver, Canada.

Kallmeyer, Laura and Maribel Romero. 2008. Scope and situation binding in ltag using semantic unification. *Research on Language and Computation*, 6(1):3–52.

Kaplan, Ronald and Paula Newman. 1997. Lexical Resource Reconciliation in the Xerox Linguistic Environment. In *Proceedings of the ACL Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, pages 54–61, Madrid, Spain. Association for Computational Linguistics.

Kinyon, Alexandra. 2000. Hypertags. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING'00)*, pages 446–452, Saarbruecken, Germany.

Kinyon, Alexandra, Owen Rambow, Tatjana Scheffler, SinWon Yoon, and Aravind K. Joshi. 2006. The metagrammar goes multilingual: A cross-linguistic look at the v2-phenomenon. In *Proceedings of the Eighth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 17–24, Sydney, Australia, July. Association for Computational Linguistics.

Le Hong, Phuong, Thi-Min-Huyen N'Guyen, and Azim Roussanaly. 2008. A metagrammar for Vietnamese. In *proceedings of the 9th International workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+9)*, Tübingen, Germany.

Lehmann, Sabine, 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 *Proceedings of the 16th International Conference on Computational Linguistics (COLING'96)*, pages 711–716, Copenhagen, Denmark.

Lichte, Timm. 2007. An MCTAG with Tuples for Coherent Constructions in German. In *Proceedings of the 12th Conference on Formal Grammar (FG 2007)*, Dublin, Ireland. 12 pages.

Muskens, Reinhard and Emiel Krahmer. 1998. Description Theory, LTAGs and Underspecified Semantics. In *Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 112–115, Philadelphia, PA. Institute for Research in Cognitive Science.

Parmentier, Yannick. 2007. *SemTAG : une plate-forme pour le calcul sémantique à partir de Grammaires d'Arbres Adjoints*. Ph.D. thesis, Université Henri Poincaré - Nancy.

Parmentier, Yannick, Laura Kallmeyer, Timm Lichte, and Wolfgang Maier. 2007. XMG: eXtending MetaGrammars to MCTAG. In *Proceedings of the Workshop on High-Level Syntactic Formalims, 14th Conference on Natural Language Processing (TALN'2007)*, pages 473–482, Toulouse, France.

Pereira, Fernando and David Warren. 1980. Definite clause grammars for language analysis —a survey of the formalism and a comparison to augmented transition networks. *Artificial Intelligence*, 13:231–278.

Perlmutter, David. 1970. Surface structure constraints in syntax. *Linguistic Inquiry*, 1:187–255.

Perrier, Guy. 2000. Interaction Grammars. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, pages 600–606, Saarbruecken, Germany.

Perrier, Guy. 2007. A French Interaction Grammar. In *proceedings of the 6th Conference on Recent Advances in Natural Language Processing (RANLP 2007)*, pages 463–467, Borovets, Bulgaria, September.

Prolo, Carlos A. 2002. Generating the XTAG English Grammar Using Metarules. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING'2002)*, pages 814–820, Taipei, Taiwan.

Rambow, Owen, K. Vijay-Shanker, and David Weir. 1995. D-tree Grammars. In *Proceedings of the 33th Meeting of the Association for Computational Linguistics*, pages 151–158, Cambridge, MA.

Rogers, James and K. Vijay-Shanker. 1994. Obtaining trees from their descriptions: An application to tree-adjoining grammars. *Computational Intelligence, 10:401–421*.

Shieber, Stuart M. 1984. The design of a computer language for linguistic information. In *Proceedings of the Tenth International Conference on Computational Linguistics*, pages 362–366, Stanford University, Stanford, CA, July2–6.

Van Roy, Peter. 1990. Extended dcg notation: A tool for applicative programming in prolog. Technical report, Technical Report UCB/CSD 90/583, UC Berkeley.

Vijay-Shanker, K. and Aravind K. Joshi. 1988. Feature Structures Based Tree Adjoining Grammars. In *Proceedings of the 12th Conference on Computational linguistics (COLING'88)*, pages 714–719, Budapest, Hungary.

Vijay-Shanker, K. and Yves Schabes. 1992. Structure sharing in lexicalized tree adjoining grammars. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING'92)*, pages 205–212, Nantes, France.

Villemonte de La Clergerie, Éric. 2005. DyALog: a tabular logic programming based environment for NLP. In *Proceedings of 2nd International Workshop on Constraint Solving and Language Processing (CSLP'05)*, pages 18–33, Barcelona, Spain, October.

Villemonte de la Clergerie, Éric. 2010. Building factorized TAGs with meta-grammars. In *Proceedings of the 10th international workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+10)*, pages 111–118, New Haven, USA.

Weir, David J. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. thesis, University of Pennsylvania.

Xia, Fei. 2001. *Automatic Grammar Generation from two Different Perspectives*. Ph.D. thesis, University of Pennsylvania.

Xia, Fei, Martha Palmer, and K. Vijay-Shanker. 1999. Toward semi-automating grammar development. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium(NLPRS-99)*, Beijing, China.

Xia, Fei, Martha Palmer, and K. Vijay-Shanker. 2005. Automatically Generating Tree Adjoining Grammars from Abstract Specifications. *Journal of Computational Intelligence*, 21(3):246–287.

Xia, Fei, Martha Palmer, and K. Vijay-Shanker, 2010. *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*, chapter Developing Tree-Adjoining Grammars with Lexical Descriptions, pages 73–110. MIT Press.

Xia, Fei, Martha Palmer, K. Vijay-Shanker, and Joseph Rosenzweig. 1998. Consistent Grammar Development Using Partial-Tree Descriptions for LTAGs. In *Proceedings of the 4th International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 1998)*, pages 180–183, Philadelphia, PA.

XTAG Research Group. 2001. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.