A constraint driven metagrammar

Joseph Le Roux
LORIA
Institut National
Polytechnique de Lorraine
615, Rue du Jardin Botanique
54 600 Villers-Lès-Nancy
France
leroux@loria.fr

Benoît Crabbé
HCRC / ICCS
University of Edinburgh
2 Buccleuch Place
EH8 9LW,
Edinburgh, Scotland
bcrabbe@inf.ed.ac.uk

Yannick Parmentier
INRIA / LORIA
Université Henri Poincaré
615, Rue du Jardin Botanique
54 600 Villers-Lès-Nancy
France
parmenti@loria.fr

Abstract

We present an operational framework allowing to express a large scale Tree Adjoining Grammar (TAG) by using higher level operational constraints on tree descriptions. These constraints first meant to guarantee the well formedness of the grammatical units may also be viewed as a way to put model theoretic syntax at work through an efficient offline grammatical compilation process. Our strategy preserves TAG formal properties, hence ensures a reasonable processing efficiency.

1 Introduction

This paper is concerned with the semi-automatic grammar development of real-scale grammars. For natural language syntax, lexicalised TAGs are made of thousands of trees, carrying an extreme structural redundancy. Their development and their maintenance is known to be cumbersome as the size of the grammar raises significantly.

To counter the lack of generalisations inherent to strong lexicalisation, various proposals for semi-automatic grammar development have been carried out: lexical rules or meta-rules (Becker, 2000) and metagrammars: (Candito, 1999; Gaiffe et al., 2002; Xia, 2001). The aim of these frameworks is twofold: expressing general facts about the grammar of a language and factorising the information to avoid redundancy.

The metagrammar path adopts a different perspective from the lexical rule based grammar development: instead of describing how a derived tree is different from a canonical one, grammatical description mainly consists of combining fragmentary tree descriptions or building blocks.

The paper is structured as follows. We start in section 2 by providing motivations and background information on the framework we are using. Section 3 shows that the metagrammar framework may be viewed as an offline system allowing to express high level well-formedness constraints on elementary grammatical structures while preserving TAG computational and formal properties. Section 4 shows how to implement efficiently this constraint-based approach with logic programming techniques and finally section 5 provides an idea of the performance of the implemented system.

2 eXtensible MetaGrammar (XMG)

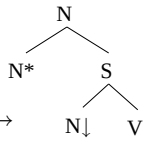
By opposition to other metagrammatical frameworks, XMG (Duchier et al., 2004) uses an expressive though simple language, enabling a monotonic description of a real scale grammar. Monotonicity is important because it means that the order of application of the different operations does not matter. This is the major drawback of lexical-rule systems. Moreover, (Crabb´e, 2005b) shows that it is sufficiently expressive to implement conveniently a core TAG for French.

XMG allows the grammar writer to manipulate tree descriptions through a control language. The intuition behind is that a metagrammatical language needs to provide means to describe syntactic information along two methodological axis (Crabb´e, 2005b): *structure sharing* and *alternatives*. Structure sharing is the axis dedicated to express factorisation in the grammar, whereas alternatives allow to express regular alternation relationships such as alternatives between the representation of a canonical nominal subject and its interrogative representation, or between an active

and a passive verb form[1].

Building on this intuition the XMG language allows the user to name partial tree descriptions within classes. The name of the class can be manipulated afterwards. For instance the following tree descriptions on the right of the arrow are associated with the names stated on the left of the arrow[2]:

(1) a. *CanonicalSubject* → 

b. *RelativisedSubject* → 

c. *VerbalForm* → 

Naming is the main device that allows the grammar writer to express and to take advantage of the structure sharing axis mentioned above. Indeed class names can be reused in other descriptions. Thus names can also be used to describe alternatives. To express, in our simplified example, that a *Subject* is an abstract way to name a *RelativisedSubject* or a *CanonicalSubject*, we use a choice operator ($\vee$) as illustrated below:

(2) Subject → CanonicalSubject
$\vee$ RelativisedSubject

Disjunction (non-deterministic choice) is the device provided by the language to express the methodological axis of alternatives.

Finally, names can be given to class combinations. To express the composition of two tree descriptions in the language, we use the $\wedge$ operator. Thus we can say that an *IntransitiveVerb* is made by the composition of a *Subject* and a *VerbalForm* as follows:

(3) IntransitiveVerb → Subject $\wedge$ VerbalForm

Given these 3 primitives, the control language is naturally interpreted as a context free grammar whose terminals are tree descriptions and where our composition plays the role of concatenation. This abstract grammar or metagrammar is further restricted to be non recursive in order to ensure that the generated TAG is finite.

Provided the axiom *IntransitiveVerb*, an interpreter for this language generates non deterministically all the sentences of the grammar[3] underlying a grammatical description. Thus in our current example the two sentences generated are those depicted on the left hand side of the arrows in Figure 1. On the right hand side of the arrow is depicted the result of the composition of the tree descriptions.

It remains to make clear what is actually this composition. The grammatical classes may contain information on tree descriptions and/or express composition of descriptions stated in other classes. Tree descriptions take their inspiration from the logic described in (Rogers and Vijay-Shanker, 1994). Its syntax is the following:

$$\begin{array}{lll} \textit{Description} & ::= & x \rightarrow y \mid x \rightarrow^* y \mid \\ & & x \prec y \mid x \prec^* y \mid \\ & & x[f{:}E] \end{array}$$

where x, y are node variables, $\rightarrow$ the dominance relation, $\prec$ the precedence relation, * denoting the reflexive transitive closure of a relation. The last line associates x with a feature f whose value is the result of evaluating expression E.

Tree descriptions are interpreted as finite linear ordered trees being the minimal models of the description.

Using tree descriptions, the above mentioned operation of tree "composition" breaks down to a conjunction of formulas where variables of each conjunct are in first approximation renamed to avoid name collisions. Renaming is a crucial difference with previous approaches to metagrammar (Candito, 1999; Xia, 2001) where the user had to manage explicitly a "global namespace". Here a specific attention is given to namespace management, because this was a bottleneck for real scale

[1]The passive is a semi-regular alternation, many transitive verbs do not passivise. Our system presupposes a classical architecture for the computational representation of Tree Adjoining Grammars such as XTAG, where means to express such exceptions during the anchoring process are well-known. In what follows, we therefore consider only tree templates (or tree schematas) as our working units. Finally the trees depicted in this paper take their inspiration from the grammar described by (Abeill´e, 2002).

[2]To represent the tree descriptions mentioned in this paper, we use a graphical notation. Immediate dominance is depicted with a straight line and precedence follows the graphical order. Note that nodes are decorated with their labels only, ignoring the names of the variables denoting them. Note also that we use only the reflexive transitive closure of precedence between sibling nodes and it is explicitly stated with the symbol $\prec^*$.

[3]Understood as compositions of tree fragments.

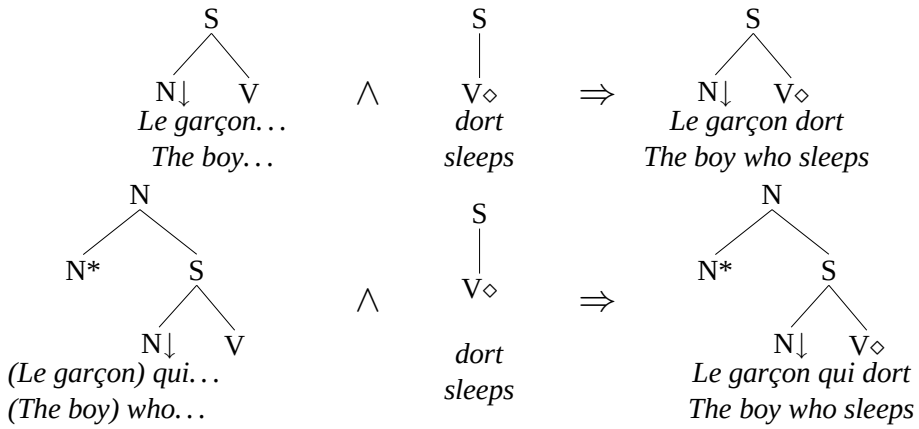


Figure 1: Interpretation of a grammatical description

grammar design. More precisely each class has its own namespace of identifiers and namespace merging can be triggered when a class combination occurs. This merging relies on a fine-grained import/export mechanism.

In addition to conjunction and disjunction, XMG is augmented with syntactic sugar to offer some of the features other metagrammatical formalisms propose. For instance, inheritance of classes is not built-in in the core language but is realised through conjunction and namespace import. Of course, this restricts users to monotonic inheritance (specialisation) but it seems to be sufficient for most linguists.

3 Constraining admissible structures

XMG has been tested against the development of a large scale French Grammar (Crabb´e, 2005a). To ease practical grammatical development we have added several augmentations to the common tree description language presented so far in order to further restrict the class of admissible structures generated by the metagrammar.

Further constraining the structures generated by a grammar is a common practice in computational linguistics. For instance a Lexical Functional Grammar (Bresnan and Kaplan, 1982) further restricts the structures generated by the grammar by means of a functional uniqueness and a functional completeness principles. These constraints further restricts the class of admissible structures generated by an LFG grammar to verify valency conditions.

For TAG and in a theoretical context, (Frank, 2002) states a set of such well formedness principles that contribute to formulate a TAG theory within a minimalist framework. In what remains we describe operational constraints of this kind that further restrict the admissibility of the structure generated by the metagrammar. By contrast with the principles stated by (Frank, 2002), we do not make any theoretical claim, instead we are stating operational constraints that have been found useful in practical grammar development.

However as already noted by (Frank, 2002) and by opposition to an LFG framework where constraints apply to the syntactic structure of a sentence as a whole, we formulate here constraints on the well-formedness of TAG elementary trees. In other words these constraints apply to units that define themselves their own global domain of locality. In this case, it means that we can safely ignore locality issues while formulating our constraints. This is theoretically weaker than formulating constraints on the whole sentential structure but this framework allows us to generate common TAG units, preserving the formal and computational properties of TAG.

We formulate this constraint driven framework by specifying conditions on model admissibility. Methodologically the constraints used in the development of the French TAG can be classified in four categories: formal constraints, operational constraints, language dependent constraints and theoretical principles.

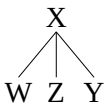
First the *formal constraints* are those constraining the trees generated by the model builder to be regular TAG trees. These constraints require the trees to be linear ordered trees with appropriate decorations : each node has a category label, leaf nodes are either terminal, foot or substitution, there is at most one foot node, the category of the foot note is identical to that of the root node, each tree has at least one leaf node which is an anchor.

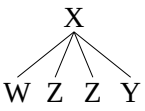
It is worth noting here that using a different set of formal constraints may change the target formalism. Indeed XMG provides a different set of formal constraints (not detailed here) that allow to generate elementary units for another formalism, namely Interaction Grammars.

The second kind of constraint is a single *operational constraint* dubbed the colouration constraint. We found it convenient in the course of grammar development. It consists of associating colour-based polarities to the nodes to ensure a proper combination of the fragmentary tree descriptions stated within classes. Since in our framework descriptions stated in two different classes are renamed before being conjoined, given a formula being the conjunction of the two following tree descriptions :

(4) 

both the following trees are valid models of that formula:

(5) (a) 

(b) 

In the context of grammar development, however, only (a) is regarded as a desired model. To rule out (b) (Candito, 1999; Xia, 2001) use a naming convention that can be viewed as follows[4]: they assign a name to every node of the tree description. Both further constrain model admissibility by enforcing the identity of the interpretation of two variables associated to the same name. Thus the description stated in their systems can be exemplified as follows:

(6) 

Though solving the initial formal problem, this design choice creates two additional complications: (1) it constrains the grammar writer to manually manage a global naming, entailing obvious problems as the size of the grammatical description grows and (2) it prevents the user to reuse several times the same class in a composition. This case is a real issue in the context of grammatical development since a grammar writer willing to describe a ditransitive context with two prepositional phrases cannot reuse two times a fragment describing such a PP since the naming constraint will identify them.

To solve these problems we use a colouration constraint. This constraint associates unary properties, colours, to every node of the descriptions. A colour is taken among the set *red(*$\bullet_R$*)*, *black(*$\bullet_B$*)*, *white (*$\circ_W$*)*. A valid model is a model in which every node is coloured either in red or black. Two variables in the description interpreted by the same node have their colours merged following the table given in Figure 2.

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

Figure 2: Colour identification rules.

The table indicates the resulting colour after a merge. The $\bot$ symbol indicates that this two colours cannot be merged and hence two nodes labelled with these colours cannot be merged. Note that the table is designed to ensure that merging is not a procedural operation.

The idea behind colouration is that of saturating the tree description. The colour white represents the non saturation or the *need* of a node to be combined with a *resource*, represented by the colour black. Black nodes need not necessarily be combined with other nodes. Red is the colour used to label nodes that cannot be merged with any other node. A sample tree description with coloured node is as follows:

(7) 

Colours contribute to rule out the (b) case and remove the grammar writer the burden of managing manually a “global namespace”.

The third category of constraints are *language dependent constraints*. In the case of French, such constraints are clitic ordering, islands constraints, etc. We illustrate these constraints with clitic ordering in French. In French clitics are non tonic particles with two specific properties already identified by (Perlmutter, 1970): first they appear in front of the verb in a fixed order according to their rank (8a-8b) and second two different clitics in front of the verb cannot have the same rank (8c). For instance the clitics *le*, *la* have the rank 3 and *lui* the rank *4*.

[4]They actually use a different formal representation that does not affect the present discussion.

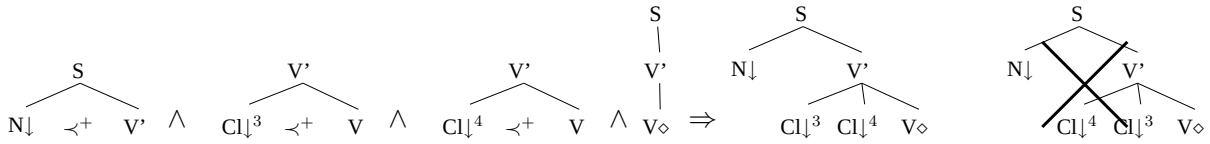


Figure 3: Clitic ordering

(8) 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

In the French grammar of (Crabb´e, 2005a) trees with clitics are generated with the fragments illustrated on the left of the arrow in Figure 3[5]. As illustrated on the right of the arrow, the composition may generate ill-formed trees. To rule them out we formulate a clitic ordering constraint. Each variable labelled with a clitic category is also labelled with a property, an integer representing its rank. The constraint stipulates that sibling nodes labelled with a rank have to be linearly ordered according to the order defined over integers.

Overall language dependent constraints handle cases where the information independently specified in different fragments may interact. These interactions are a counterpart in a metagrammar to the interactions between independently described lexical rules in a lexical rule based system. Assuming independent lexical rules moving canonical arguments (NP or PP) to their clitic position, lexical rules fall short for capturing the relative ordering among clitics[6].

A fourth category of constraints, not implemented in our system so far are obviously the *language independent principles* defining the theory underlying the grammar. Such constraints could involve for instance a Principle of Predicate Argument Coocurrency (PPAC) or even the set of minimalist principles described by (Frank, 2002).

4 Efficient implementation

We describe now the implementation of our metagrammatical framework. In particular, we will focus on the implementation of the constraints discussed above within XMG.

As mentioned above, a metagrammar corresponds to a reduced description of the grammar. In our case, this description consists of tree fragments combined either conjunctively or disjunctively. These combinations are expressed using a language close to the *Definite Clause Grammar* formalism (Pereira and Warren, 1980), except that partial tree descriptions are used as terminal symbols. In this context, a metagrammar can be reduced to a logic program whose execution will lead to the computation of the trees of the grammar.

To perform this execution, a compiler for our metagrammatical language has been implemented. This compilation is a 3-step process as shown in Figure 4.

First, the metagrammar is compiled into instructions for a specific virtual machine inspired by the Warren's Abstract Machine (Ait-Kaci, 1991). These instructions correspond to the unfolding of the relations[7] contained in the tree descriptions of the metagrammar.

Then, the virtual machine performs unifications of structures meant to refer to corresponding information within fragments (*e.g.* two nodes, two feature structures ...). Note that the XMG's virtual machine uses the *structure sharing* technique for memory management, *i.e.* data are represented by a pair *pattern – environment* in which to interpret it. The consequences are that (a) we save memory when compiling the metagrammar, and (b) we have to perform pointer dereferencing during unification. Even if the latter is time-consuming, it remains more efficient than *structure copying* as we have to possibly deal with a certain amount of tree descriptions.

Eventually, as a result of this instruction processing by the virtual machine, we obtain poten-

[5] Colours are omitted.

[6] This observation was already made by (Perlmutter, 1970) in a generative grammar framework where clitics where assumed to be moved by transformations.

[7] These relations are either dominance or precedence between node variables, or their reflexive transitive closure, or the labelling of node variable with feature structures.

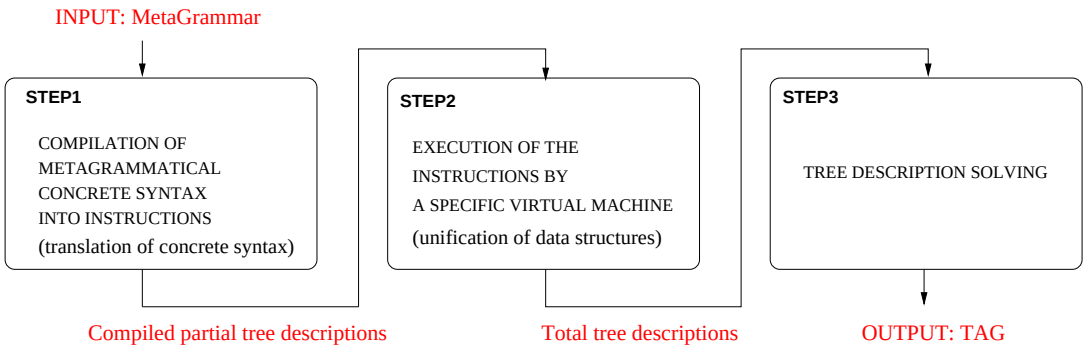


Figure 4: Metagrammar compilation.

tially total tree descriptions, that have to be solved in order to produce the expected TAG.

Now, we will introduce XMG's tree description solver and show that it is naturally designed to process efficiently the higher level constraints mentioned above. In particular, we will see that the description solver has been designed to be easily extended with additional parametric admissibility constraints.

4.1 Tree descriptions solving

To find the minimal models corresponding to the total tree descriptions obtained by accumulating fragmentary tree descriptions, we use a tree description solver. This solver has been developed in the *Constraint Programming* paradigm using the constraint satisfaction approach of (Duchier and Niehren, 2000). The idea is to translate relations between node variables into constraints over sets of integers.

Basically, we refer to a node of the input description in terms of the nodes being equals, above, below, or on its side (see Figure 5). More precisely, we associate each node of the description with an integer, then our reference to a node corresponds to a tuple containing sets of nodes (*i.e.* sets of integers).

As a first approximation, let us imagine that we refer to a node x in a model by means of a 5-tuple $N_x^i = (Eq, Up, Down, Left, Right)$ where i is an integer associated with x and *Eq* (respectively *Up*, *Down*, *Left*, *Right*) denotes the set of nodes[8] in the description which are equal, (respectively above, below, left, and right) of x.

Then we can convert the relations between nodes of our description language into constraints on sets of integer.

[8] *I.e. integers.*

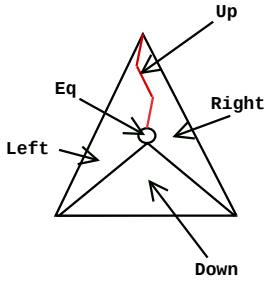


Figure 5: Node representation.

For instance, if we consider 2 nodes x and y of the description. Assuming we associate x with the integer i and y with j, we can translate the dominance relation $x \rightarrow y$ the following way[9]:

$$N_x^i \rightarrow N_y^j \equiv \begin{array}{l}[N_{x.EqUp}^i \subseteq N_{y.Up}^j \wedge N_{x.Down}^i \supseteq N_{y.EqDown}^j \\ \wedge N_{x.Left}^i \subseteq N_{y.Left}^j \wedge N_{x.Right}^i \subseteq N_{y.Right}^j]\end{array}$$

This means that if the node[10] x strictly dominates y in the input description, then (i) the set of nodes that are above or equal x in a valid model is included in the set of those that are strictly above y *and* (ii) the dual holds for the nodes that are above *and* (iii) the set of nodes that are on the left of y is included in the set of those that are on the left of x *and* (iv) similarly for the right part.

Once the constraints framework is settled, we can search for the solutions to our problem, *i.e.* the variable assignments for each of the sets of integers used to refer to the nodes of the input description. This search is performed by associating with each pair of nodes (x, y) of the input description a choice variable denoting the mutually exclusive relations[11] between these two nodes. Then

[9] $N_{x.EqUp}^i$ corresponds to the disjoint union of $N_{x.Eq}^i$ and $N_{x.Up}^i$, similarly for $N_{x.EqDown}^j$ with $N_{x.Eq}^i$ and $N_{x.Down}^i$.

[10] One should read the node denoted by the variable x.

[11] Either x equals y, x dominates y, y dominates x, x precedes y or y precedes x.

we use a search strategy to explore the consistent assignments to these choices variables (and the associated assignments for sets of integers referring to nodes)[12]. Note that the strategy used in XMG is a *first-fail* strategy which leads to very good results (see section 5 below). The implementation of this solver has been done using the constraint programming support of the Mozart Programming System (The Oz-Mozart Board, 2005).

4.2 Extension to higher-level constraints solving

An important feature of our approach is that this system of constraints over integer sets can be extended so that we not only ensure tree well-formedness of the outputted trees, but also the respect of linguistic properties such as the uniqueness of clitics in French, etc.

The idea is that if we extend adequately our node representation, we can find additional constraints that reflects the syntactic constraints we want to express.

Clitic uniqueness For instance, let us consider the *clitic uniqueness* constraint introduced above. We want 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 category *clitic*[13]). This can be done by introducing, for each node x *of the description*, a boolean variable p_x indicating whether the node denoting x *in the model* has this property or not. Then, if we call $\mathcal{V}_p^\phi$ the set of integers referring to nodes having the property p *in a model*, we have:

$$p_x \equiv (N^i_{x.Eq} \cap \mathcal{V}_p^\phi) \neq \emptyset$$

Finally, if we represent the true value with the integer 1 and false with 0, we can sum the p_x for each x *in the model*. When this sum gets greater than 1, we can consider that we are not building a valid model.

Colouration constraint Another example of the constraints introduced in section 3 is colouration. Colouration represents operational constraints whose effect is to control tree fragment combination. The idea is to label nodes with a colour between red, black and white. Then, during description solving, nodes are identified according to the rules given previously (see Figure 2).

That is, red nodes are not identified with any other node, white nodes can be identified with a black one. Black nodes are not identified with each other. A valid model in this context is a saturated tree, *i.e.* where nodes are either black (possibly resulting from identifications) or red. In other words, for every node in the model, there is at most one red or black node with which it has been identified. The implementation of such a constraint is done the following way. First, the tuples representing nodes are extended by adding a integer field *RB* referring to the red or black node with which the node has been identified. Then, considering the following sets of integers: $\mathcal{V}_\text{R}$, $\mathcal{V}_\text{B}$, $\mathcal{V}_\text{W}$ respectively containing the integers referring to red, black and white nodes in the input description, the following constraints hold:

$$\begin{aligned}
x \in \mathcal{V}_\text{R} \quad &\Rightarrow \quad N^i_{x.RB} = i \wedge\ N^i_{x.Eq} = \{i\} \quad (a)\\
x \in \mathcal{V}_\text{B} \quad &\Rightarrow \quad N^i_{x.RB} = i \quad (b)\\
x \in \mathcal{V}_\text{W} \quad &\Rightarrow \quad N^i_{x.RB} \in \mathcal{V}_\text{B}^\phi \quad (c)
\end{aligned}$$

where $\mathcal{V}_\text{B}^\phi$ represents the black nodes in a model, *i.e.* $\mathcal{V}_\text{B}^\phi = \mathcal{V}^\phi \cap \mathcal{V}_\text{B}$. (a) expresses the fact that for red nodes, $N^i_{x.RB}$ is the integer i associated with x itself, and $N^i_{x.Eq}$ is a set only containing i. (b) means that for black nodes, we have that $N^i_{x.RB}$ is also the integer i denoting x itself, but we cannot say anything about $N^i_{x.Eq}$. Eventually (c) means that whites nodes have to be identified with a black one.

Thus, we have seen that *Constraint Programming* offers an efficient and relatively natural way of representing syntactic constraints, as "all" that has to be done is to find an adequate node representation in terms of sets of nodes, then declare the constraints associated with these sets, and finally use a search strategy to compute the solutions.

5 Some features

There are two points worth considering here: (i) the usability of the formalism to describe a real scale grammar with a high factorisation, and (ii) the efficiency of the implementation in terms of time and memory use.

Concerning the first point, XMG has been used successfully to compute a TAG having more than 6,000 trees from a description containing 293

[12] More information about the use of such choice variables is given in (Duchier, 1999)

[13] In fact, the uniqueness concerns the rank of the clitics, see (Crabb´e, 2005b), §9.6.3.

classes[14]. Moreover, this description has been designed relatively quickly as the description language is intuitive as advocated in (Crabb´e, 2005a).

Concerning the efficiency of the system, the compilation of this TAG with more than 6,000 trees takes about 15 min with a P4 processor 2.6 GHz and 1 GB RAM. Note that compared with the compilation time of previous approaches (Candito, 1999; Gaiffe et al., 2002) (with the latter, a TAG of 3,000 trees was compiled in about an hour), these results are quite encouraging.

Eventually, XMG is released under the terms of the GPL-like CeCILL license[15] and can be freely downloaded at `http://sourcesup.cru.fr/xmg`.

6 Conclusion

Unlike previous approaches, the description language implemented by XMG is fully declarative, hence allowing to reuse efficient techniques borrowed to Logic Programming. The system has been used successfully to produce core TAG (Crabb´e, 2005b) and *Interaction Grammar* (Perrier, 2003) for French along with a core French TAG augmented with semantics (Gardent, 2006).

This paper shows that the metagrammar can be used to put model theoretic syntax at work while preserving reasonably efficient processing properties. The strategy used here builds on constraining offline a TAG whose units are elementary trees The other option is to formulate constraints applied on-line, in the course of parsing, applying on the whole syntactic structure. In a dependency framework, XDG followed this path (Debusmann et al., 2004), however it remains unknown to us whether this approach remains computationally tractable for parsing with real scale grammars.

References

A. Abeill´e. 2002. Une grammaire ´electronique du franais. CNRS Editions, Paris.

H. Ait-Kaci. 1991. Warren's abstract machine: A tutorial reconstruction. In K. Furukawa, editor, *Proc. of the Eighth International Conference of Logic Programming*. MIT Press, Cambridge, MA.

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

Joan Bresnan and Ronal M. Kaplan. 1982. *The Mental Representation of Grammatical Relations*. The MIT Press, Cambridge MA.

M.H. Candito. 1999. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées : application au franç ais et à l'italien*. Ph.D. thesis, Universit´e Paris 7.

B. Crabb´e. 2005a. Grammatical development with XMG. Proceedings of the Fifth International Conference on Logical Aspects of Computational Linguistics (LACL05).

B. Crabb´e. 2005b. *Représentation informatique de grammaires fortement lexicalisées : Application à la grammaire d'arbres adjoints*. Ph.D. thesis, Universit´e Nancy 2.

R. Debusmann, D. Duchier, and G.-J. M. Kruijff. 2004. Extensible dependency grammar: A new methodology. In *Proceedings of the COLING 2004 Workshop on Recent Advances in Dependency Grammar*, Geneva/SUI.

D. Duchier and J. Niehren. 2000. Dominance constraints with set operators. In *Proceedings of CL2000*, volume 1861 of *Lecture Notes in Computer Science*, pages 326–341. Springer.

D. Duchier, J. Le Roux, and Y. Parmentier. 2004. The Metagrammar Compiler: An NLP Application with a Multi-paradigm Architecture. In *2nd International Mozart/Oz Conference (MOZ'2004)*, Charleroi.

D. Duchier. 1999. Set constraints in computational linguistics - solving tree descriptions. In *Workshop on Declarative Programming with Sets (DPS'99), Paris, pp. 91 - 98*.

Robert Frank. 2002. *Phrase Structure Composition and Syntactic Dependencies*. MIT Press, Boston.

B. Gaiffe, B. Crabb´e, and A. Roussanaly. 2002. A new metagrammar compiler. In *Proceedings of TAG+6, Venice*.

C. Gardent. 2006. Int´egration d'une dimension s´emantique dans les grammaires d'arbres adjoints. In *Actes de La 13ème édition de la conférence sur le TALN (TALN 2006)*.

F. Pereira and D. 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.

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

Guy Perrier. 2003. Les grammaires d'interaction. HDR en informatique, Universit´e Nancy 2.

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

The Oz-Mozart Board. 2005. The Oz-Mozart Programming System. http://www.mozart-oz.org.

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

[14] I.e. tree fragments or conjunction / disjunction of fragments

[15] More information about this license at `http://www.cecill.info/index.en.html`.