XMG : Un Compilateur de Méta-Grammaires Extensible

Denys Duchier, Joseph Le Roux, Yannick Parmentier

TALN 2005, Dourdan, 6–10 juin 2005

XMG : Un Compilateur de Méta-Grammaires Extensible*

Denys Duchier (1), Joseph Le Roux (2), Yannick Parmentier (2)
(1) LIFL - UMR CNRS 8022- Université des Sciences et Technologies de Lille
Bâtiment M3 59655 Villeneuve d'Ascq Cédex
duchier@lifl.fr
(2) INRIA / LORIA - Institut National Polytechnique de Lorraine - Université Henri Poincaré Nancy 1
615, rue du Jardin Botanique - 54 600 Villers-Lès-Nancy
{leroux,parmenti}@loria.fr

Mots-clefs : Grammaires, compilation, ressources linguistiques, Grammaires d'Arbres Adjoints, Grammaires d'Interaction

Keywords: Grammars, compilation, linguistic resources, Tree Adjoining Grammars, Interaction Grammars

Résumé Dans cet article, nous présentons un outil permettant de produire automatiquement des ressources linguistiques, en l'occurence des grammaires. Cet outil se caractérise par son extensibilité, tant du point de vue des formalismes grammaticaux supportés (grammaires d'arbres adjoints et grammaires d'interaction à l'heure actuelle), que de son architecture modulaire, qui facilite l'intégration de nouveaux modules ayant pour but de vérifier la validité des structures produites. En outre, cet outil offre un support adapté au développement de grammaires à portée sémantique.

Abstract In this paper, we introduce a new tool for automatic generation of linguistic resources such as grammars. This tool's main feature consists of its extensibility from different points of view. On top of supporting several grammatical formalisms (Tree Adjoining Grammars and Interaction Grammars for now), it has a modular architecture which eases the integration of modules dedicated to the checking of the output structures. Furthermore, this tool offers adapted support to the development of grammars with semantic information.

* Nous tenons à remercier Benoît Crabbé, Claire Gardent, Guy Perrier et Eric De La Clergerie pour les nombreuses discussions qui nous ont aidées dans le cadre de ce travail.

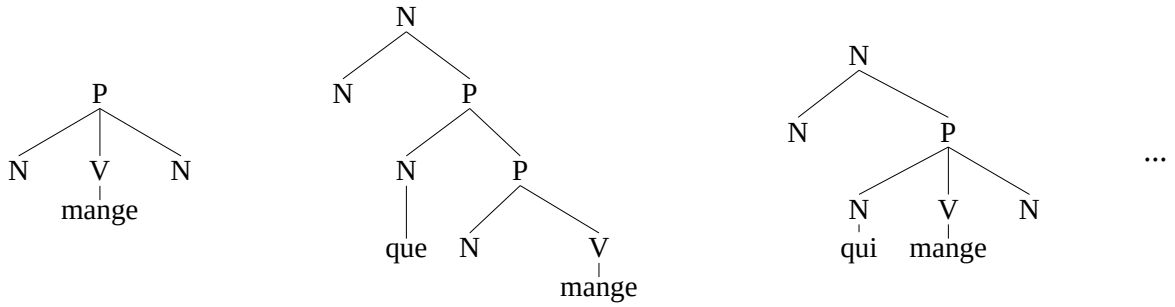
1 La production automatique de grammaires

Dans le cadre du développement d'applications de TALN (analyseurs syntaxiques par exemple), un certain nombre de ressources linguistiques est nécessaire, parmi lesquelles des grammaires. Nous nous intéressons ici aux grammaires fortement lexicalisées associant à chaque mot un ensemble de structures syntaxiques décrivant son comportement dans chaque emploi.

1.1 Le besoin de production automatique

Par opposition aux grammaires de règles, les grammaires fortement lexicalisées utilisées en linguistique informatique, comme les grammaires d'arbres adjoints lexicalisées (TAG), les grammaires d'interaction (IG) ou les grammaires catégorielles reportent la connaissance de la langue dans le lexique. Un tel lexique peut être vu comme une fonction associant à un mot de la langue l'ensemble des structures syntaxiques représentant ses usages. Pour avoir une bonne couverture, il est nécessaire que le lexique associe à chaque mot un maximum de structures. Par exemple pour analyser les expressions ci-dessous, le lexique doit associer au verbe *mange* (en TAG) les structures suivantes :

Jean mange la pomme. *La pomme que Jean mange ...* *Jean qui mange une pomme.* ...



Cette taille de lexique ne va pas sans problème[1]. Comme les règles syntaxiques sont *éclatées* à travers le lexique, l'analyse d'un nouveau phénomène linguistique ou, pire, sa révision peuvent avoir de graves conséquences sur la cohérence de la grammaire. Si les premiers lexiques à large couverture furent écrits à la main, leur génération automatique devient de plus en plus pressante, de manière à (1) pouvoir garantir la cohérence de l'ensemble et (2) pouvoir réviser une grammaire rapidement, même quand le lexique devient important. Nous avons développé un formalisme, dont l'implantation est XMG, qui, à partir d'une description métagrammaticale concise génère toutes les structures associées aux mots du lexique. Notre approche se veut la plus indépendante possible des formalismes grammaticaux et capable de gérer simultanément les aspects syntaxiques et sémantiques du lexique.

1.2 Principes de la production automatique

Le lexique est extrêmement redondant : d'une part, une même structure syntaxique peut être associée à plusieurs entrées lexicales (*e.g.* les verbes transitifs auront un grand nombre de structures syntaxiques en commun) et d'autre part les différentes structures lexicales partagent des

[1] Les problèmes purement informatiques ne sont pas évoqués ici.

fragments communs importants (*e.g.* le fragment sujet-verbe se retrouve de très nombreuses fois). L'idée qui s'est imposée dès les premiers travaux de génération automatique de lexique par (Candito, 1996) est de ne décrire que ces fragments élémentaires, pouvant hériter les uns des autres, puis de les combiner pour former toutes les structures syntaxiques du lexique. Ces combinaisons constituent la justification linguistique de cette approche[2]. En effet elles doivent expliquer la formation des différentes structures apparentées (par exemple la relation entre les formes actives et passives d'un même verbe). Les différents formalismes métagrammaticaux doivent donc rendre exprimables la notion de fragments réutilisables, la spécification de leur structure et la manière de les combiner pour produire des structures complètes. Par exemple, (Candito, 1996) explique les croisements par un ensemble de contraintes (de 3 types) que doivent vérifier les structures finales et (Gaiffe, Crabbé et Roussanaly, 2002) les justifie en assouplissant la notion de contrainte par une approche où besoins et ressources doivent être satisfaits. Notre approche les explique par deux actions primitives *l'accumulation* et *la composition disjonctive* auxquelles peuvent être ajoutées des contraintes (sensibilité aux ressources, comme par exemple un langage de couleur, voir section 2.2). Mais XMG se distingue des approches antérieures par trois aspects fondamentaux :

1. XMG est **multi-formalisme**, c'est à dire qu'il ne se limite pas à un formalisme syntaxique en particulier,
2. XMG est **extensible**, d'une part on peut lui ajouter des niveaux de description pour traiter de nouveaux formalismes (aussi bien syntaxiques que sémantiques), et d'autre part, on peut l'étendre en définissant des modules de contraintes additionnelles que les structures grammaticales produites doivent respecter,
3. la description métagrammaticale est vue comme un **programme logique** dont XMG est le compilateur, ce qui nous permet de réutiliser des techniques issues de la programmation logique.

1.3 L'extensibilité de XMG

XMG présente une grande originalité vis à vis des autres cadres méta-grammaticaux : il est à la fois multi-formalisme [3] (génère aussi bien des TAG que des IG), et extensible. Cette extensibilité est atteinte d'une part grâce à un langage de contrôle muni des opérations d'accumulation et de composition disjonctive qui sont générales, et ne décrivent que des relations de fragment à fragment. Ce langage est donc indépendant du formalisme grammatical cible et permet d'exprimer des combinaisons très fines. D'autre part, dans XMG, chaque fragment peut contenir un nombre arbitraire de niveaux de description (appelés aussi *dimensions*) tels que le niveau syntaxique, sémantique, etc. Ces dimensions ne sont pas complètement indépendantes puisqu'elles peuvent partager de l'information (en particulier pour l'interface syntaxe / sémantique). En outre, chaque dimension est munie de son propre langage de description qui varie selon le type d'information contenue. Par exemple, pour les TAG et les IG la dimension syntaxique est représentée par un langage de description d'arbres intégrant l'unification et où chaque nœud est équipé d'une structure de traits, tandis que pour les grammaires de dépendances extensibles (XDG) les entrées lexicales seraient des structures de traits atomiques. Enfin, XMG est extensible par l'ajout de modules spécifiques pour traiter les structures intermédiaires produites. Il existe deux types de

[2] Les approches antérieures ne s'intéressaient qu'au problème de redondance.

[3] Les approches de méta-grammaires précédentes ne permettaient que la production de grammaires TAG, bien que les travaux de (Clément et Kinyon, 2003) montrent qu'il est possible d'adapter le résultat obtenu pour générer des grammaires fonctionnelles lexicales (LFG).

modules : (1) ceux qui créent de nouvelles structures (par exemple, pour TAG, en produisant des arbres à partir de descriptions) et (2) ceux qui filtrent les structures correctes suivant des critères linguistiques (voir section 2.2).

A ce jour, la support du multi-formalisme a été validé dans XMG via TAG et IG, cependant il n'est pas encore possible de sélectionner la dimension *syntaxique* à utiliser (seule un langage de description d'arbres a été implanté) : l'extensibilité déclarative n'est pas encore atteinte.

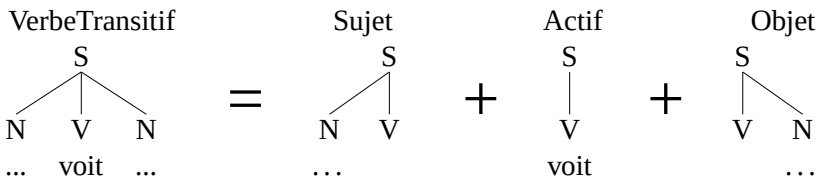
Par la suite, nous allons tout d'abord présenter le procédé de compilation de XMG, ensuite nous verrons concrètement comment produire automatiquement une grammaire TAG, enfin nous aborderons la question de l'intégration de XMG dans une chaîne de TALN.

2 Un procédé de compilation basé sur des techniques de programmation logique

Dans cette section, nous allons décrire le procédé de compilation de méta-grammaire utilisé par XMG. Plus précisément, nous allons présenter (a) le langage abstrait dont XMG est l'implantation. Notons que nous ne présentons pas ici la syntaxe *concrète* du langage (voir section 3 pour cela), mais le langage de plus bas niveau dans lequel cette syntaxe est traduite lors de la compilation. Nous présenterons également (b) l'architecture de XMG, qui par sa modularité, s'adapte aisément à différents formalismes grammaticaux.

2.1 Un langage de représentation étendu

Combinaison de fragments Nous avons vu à la section précédente que la compilation d'une méta-grammaire correspondait à la combinaison de fragments[4]. Nous pouvons définir de telles combinaisons sous forme de règles de réécriture. Par exemple, considérons l'arbre syntaxique associé à une entrée lexicale telle que *voit* (*i.e.* un verbe transitif) dans le formalisme TAG. Disposant des fragments d'arbres $Sujet$, $Actif$ et $Object$, nous pouvons réécrire l'arbre $VerbeTransitif$ comme la conjonction de ces 3 fragments :



Ce qui s'écrit également comme suit :

$$\textit{VerbeTransitif} \quad \rightarrow \quad \textit{Sujet} \wedge \textit{Actif} \wedge \textit{Objet} \tag{1}$$

Nous nous ramenons ainsi au formalisme des *grammaires de clauses définies* (DCG), dans lequel les terminaux ne seraient pas des mots mais des fragments d'arbres. La compilation d'une méta-grammaire s'identifie alors à la compilation d'un programme logique. Dans XMG, la

[4]Nous laissons provisoirement de côté la question de la contrainte des combinaisons, voir 2.2.

méta-grammaire est décrite au moyen d'un langage de représentation, qui comprend les notions d'*abstraction* (*cf* 2) et de *composition conjonctive et disjonctive* (*cf* 3).

$$\textit{Clause} \quad ::= \quad \textit{Nom} \rightarrow \textit{But} \tag{2}$$

$$\textit{But} \quad ::= \quad \textit{Description} \mid \textit{Nom} \mid \textit{But} \vee \textit{But} \mid \textit{But} \wedge \textit{But} \tag{3}$$

On remarque que XMG fournit un langage expressif incluant la disjonction et de ce fait introduit de l'indéterminisme dans la combinaison des fragments d'arbres. Ainsi, nous pouvons préciser l'exemple précédent de l'arbre associé aux verbes transitifs en spécifiant que le sujet peut être sous forme canonique *ou* sous forme relative, etc. Cela s'énonce via la règle suivante :

$$\textit{Sujet} \quad \rightarrow \quad \textit{SujetCan} \vee \textit{SujetRel} \vee \ldots \tag{4}$$

Description des fragments Les fragments dénotés par les abstractions (nommées également *classes*) de notre langage de représentation sont décrits au moyen de contraintes de dominance :

$$\textit{Description} ::= x \rightarrow y \mid x \rightarrow^{+} y \mid x \rightarrow^{*} y \mid x \prec y \mid x \prec^{+} y \mid x \prec^{*} y \mid x[f{:}E] \mid x(p{:}E) \tag{5}$$

où x, y représentent des variables de noeuds, $\rightarrow$ la dominance immédiate, $\rightarrow^{+}$ la dominance stricte, $\rightarrow^{*}$ la dominance large, $\prec$ la précédence immédiate, $\prec^{+}$ la précédence stricte, $\prec^{*}$ la précédence large, $x[f{:}E]$ l'association du trait f au noeud x et $x(p{:}E)$ l'association de la propriété p à ce même noeud x.

Ainsi, nous disposons d'un langage suffisamment expressif pour supporter les formalismes syntaxiques basés sur les descriptions d'arbres, telles que les grammaires TAG et IG.

Représentation sémantique L'extensibilité de XMG se retrouve également dans le fait qu'il offre un support adéquat à l'intégration d'une représentation sémantique dans la méta-grammaire. En effet, notre langage de représentation permet non seulement de placer dans les classes des informations syntaxiques (*e.g.* descriptions d'arbres), mais également des représentations sémantiques. A l'heure actuelle, un langage à base de structures prédicatives a été implanté :

$$\textit{Description} ::= \ell{:}p(E_1, ..., E_n) \mid \neg\ell{:}p(E_1, ..., E_n) \mid E_i \ll E_j \tag{6}$$

Ici, $\ell{:}p(E_1, ..., E_n)$ représente le prédicat p avec les arguments $E_1, .., E_n$, et étiqueté par ℓ, $\neg$ l'opérateur de négation, et $E_i \ll E_j$ la portée entre les variables sémantiques E_i et E_j.

Un tel langage peut servir, par exemple, à associer aux arbres des informations de type sémantique plate comme dans (Gardent et Kallmeyer, 2003).

En reprenant notre comparaison entre méta-grammaire et DCG se pose la question de la distinction entre information syntaxique et information sémantique. Cela est rendu possible par l'utilisation de *dimensions*, correspondant chacune à un *accumulateur* spécifique dans le formalisme des *grammaires de clauses définies étendues* (EDCG) (Van Roy, 1990).

Ainsi, la combinaison de classes (*i.e.* d'informations syntaxiques ou sémantiques) donne lieu à l'accumulation de leur contenu dans des structures dédiées. L'expression (3) vue précédemment est alors étendue en remplacant $Description$ par :

$$\textit{Dimension} \mathrel{+}= \textit{Description}$$

Ce type accumulation permet de traiter plusieurs dimensions. A l'heure actuelle, XMG dispose de 3 dimensions, notées **syn**, **sem** et **dyn**. Les deux premières représentent les dimensions syntaxique et sémantique et la dernière une dimension utilisée pour l'accumulation d'informations lexicales.

Notons que ces dimensions peuvent partager des variables logiques, ce qui permet un développement relativement naturel d'une interface syntaxe / sémantique au niveau méta-grammatical.

Portée des variables Dans les approches antérieures les noms de variables ont une portée globale à la méta-grammaire. Ce qui pose des problèmes de conflits de noms, passé un certain nombre de classes. Dans XMG, les identifiants ont par défaut une portée limitée à la clause, à laquelle est intégré un procédé d'export d'identifiants. Ainsi, il est possible de spécifier avec précision le domaine de visibilité d'une variable. Plus précisément, à chaque clause est associée une structure de traits contenant les identifiants exportés, (2) devient :

$$\textit{Clause} \quad ::= \quad \langle f_1{:}E_1, \ldots, f_n{:}E_n \rangle \Leftarrow \textit{Nom} \quad \rightarrow \quad \textit{But} \tag{7}$$

Et l'appel de classe s'accompagne de l'accès à la structure d'export (notée *Var* ici), (3) est remplacée par :

$$\textit{But} \quad ::= \quad \textit{Dim} \mathrel{+}= \textit{Description} \mid \textit{Var} \Leftarrow \textit{Nom} \mid \textit{But} \vee \textit{But} \mid \textit{But} \wedge \textit{But} \tag{8}$$

On peut accéder alors à un identifiant *X* via la notation pointée *Var.X*.

2.2 Une architecture modulaire

En section 1, nous avons présenté les caractéristiques de XMG, dont le fait qu'il supporte plusieurs formalismes syntaxiques. Cette caractéristique est fortement liée à l'architecture modulaire du compilateur.

Des modules dédiés La compilation d'une méta-grammaire se fait en plusieurs phases, dont certaines diffèrent suivant le formalisme. Chacune de ces phases est prise en charge par un module spécifique.

Actuellement XMG comporte 3 modules principaux :

a) La partie avant (*i.e.* le compilateur proprement dit) traduit la méta-grammaire en instructions exprimées dans un langage de plus bas niveau.

b) Ces instructions sont ensuite exécutées par une machine virtuelle (MV) de type *Warren's Abstract Machine* (WAM, voir (Ait-Kaci, 1991)).

Cette MV réalise l'unification des structures de données de la méta-grammaire (*i.e.* structures de traits associées aux noeuds, traits polarisés pour IG, etc), puis l'accumulation des dimensions pour une combinaison de classes donnée. En sortie de la MV, nous disposons de données accumulées dans chaque dimension, dans le cas des TAG, des descriptions d'arbres dont il faut calculer les solutions.

c) En plus de la partie avant et de la MV, qui sont communes aux formalismes des TAG et des IG, XMG intègre un module de résolution de descriptions d'arbres. Ce module est programmé sous forme d'un résolveur de contraintes (voir (Duchier et Niehren, 2000) pour une description complète du procédé).

Un résolveur extensible La modularité dans XMG est encore étendue par la programmation de modules additionnels optionnels et paramétrables. Ces modules permettent d'étendre les fonctionnalités du module de résolution pour, par exemple, contraindre les modèles produits par XMG selon des critères spécifiques, appelés aussi *principes*.

Les principes instanciables présents dans XMG actuellement sont de 3 types :

i) un principe de **couleurs**. Comme annoncé à la section 1.2, dans un contexte de développement de grammaires à large couverture par combinaison de fragments, une idée majeure consiste à contraindre les combinaisons acceptables en intégrant un système de gestion de ressources. Pour gérer les ressources en TAG, XMG intègre un langage de couleurs. Ce langage permet d'indiquer quels fragments d'arbres nécessitent quels autres fragments pour former un modèle valide (une présentation de l'emploi d'un langage de couleurs pour produire une grammaire TAG est donnée dans (Crabbé et Duchier, 2004)).

ii) un principe d'**unicité** paramétré par une propriété de noeuds. Ce principe permet de garantir la validité des solutions produites par XMG par rapport à une contrainte linguistique d'unicité telle que : *dans un arbre TAG, il ne peut y avoir deux extractions*[5].

iii) un principe de **rang** paramétré par un nombre entier, permettant de réaliser l'ordonnancement des clitiques dans les arbres TAG produits.

3 Cas d'étude : une méta-grammaire pour TAG

Voyons à présent comment écrire une méta-grammaire pour TAG avec XMG. Pour cela nous allons introduire la syntaxe *concrète* de XMG, qui est destinée à être traduite dans le langage *abstrait* présenté précédemment lors de la compilation. Notons que nous n'entrons pas ici dans les aspects du développement de grammaires à large couverture, nous nous focalisons sur un exemple simple, introductif vis à vis des possibilités offertes par XMG.

Nous allons considérer la méta-grammaire permettant de générer les arbres TAG pour les verbes transitifs à l'actif avec sujet canonique *ou* extrait *et* objet sous forme canonique (exemple 1 de la section 2.1)[6].

Les déclarations La première information que doit fournir notre méta-grammaire consiste en la spécification du résolveur utilisé (si nous en utilisons un). Ici nous allons utiliser le principe d'unicité de la fonction grammaticale sujet :

```
use unicity with (fg = suj) dims (syn)
```

Cette instruction indique à XMG que le résolveur ne doit accepter que les solutions pour lesquelles il n'y a qu'un seul noeud ayant la propriété *fg = suj* (respect du principe d'unicité).

Ensuite, nous déclarons les types de données utilisés. Ces types permettent de typer les propriétés et structures de traits qui seront associées aux noeuds dans les descriptions syntaxiques :

```
type CAT={n,v,p}
type FG={suj,obj}
property fg : FG
feature cat : CAT
```

[5] Nous adoptons cette convention dans un contexte méta-grammatical, bien que certains exemples de phrases à double extraction existent (cf (Abeillé, 2002)).

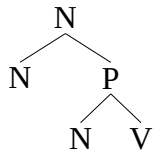
[6] Nous ne présentons pas le formalisme TAG ici, le lecteur est dirigé vers (Joshi et Schabes, 1997) pour une telle introduction.

Les classes A présent, nous pouvons définir le contenu de nos classes, *i.e.* les fragments d'arbres. Considérons le fragment correspondant au sujet canonique. Celui-ci est composé de 3 noeuds x, y et z, tels que $x \rightarrow y \wedge x \rightarrow z \wedge y \prec z$. Ce qui s'écrit comme suit dans la syntaxe de XMG ($x \rightarrow y$ correspond à `node ?X{node ?Y}`) :

```
class SujetCan
export ?X ?Z
declare ?X ?Y ?Z
{ <syn>{
        node ?X [cat = s]{
                node ?Y (fg=suj)[cat=n]
                node ?Z [cat = v]
                }
        }
}
```

La classe ObjetCan s'écrit de façon similaire. La classe SujetRel correspond à l'arbre du sujet en position relative. Le fragment d'arbre correspondant étant :

```
class SujetRel
export ?Y ?Z
declare ?X ?Y ?Z ?U ?V
{ <syn>{
        node ?U [cat = n]{
                node ?V [cat = n]
                node ?X [cat = p]{
                        node ?Y (fg=suj)[cat=n]
                        node ?Z [cat = v]
                        }
                }
        }
}
```



Il ne nous reste plus alors qu'à définir les classes Actif et VerbeTransitif. La classe Actif contient deux noeuds x, y tels que $x \rightarrow y$ et la classe VerbeTransitif se définit en suivant la règle donnée en (4) :

```
class VerbeTransitif
declare ?SU ?OB ?AC
{
       { ?SU = SujetCan[] | ?SU = SujetRel[] } ;
         ?OB = ObjetCan[] ; ?AC = Actif[]      ;
         ?SU.?X = ?OB.?X  ; ?SU.?Z = ?OB.?Y    ;
         ?SU.?X = ?AC.?X  ; ?SU.?Z = ?AC.?Y
}
```

Dans cette classe, on remarque la présence de conjonctions représentées par le symbole `;` et d'une disjonction représentée par `|`. On utilise la variable `SU` (respectivement `OB` et `AC`) pour désigner la structure de traits d'export de la classe SujetCan *ou* de la classe SujetRel (respectivement de la classe ObjetCan *et* de la classe Actif).

On remarque ici que nous procédons par égalité entre noeuds pour contraindre la combinaison des fragments d'arbres. Il s'agit du procédé de base. Pour passer au développement de méta-grammaires de taille importante, l'emploi d'un langage de couleur offre une meilleure flexibilité dans la définition des classe et permet d'alléger la gestion des noms de variables.

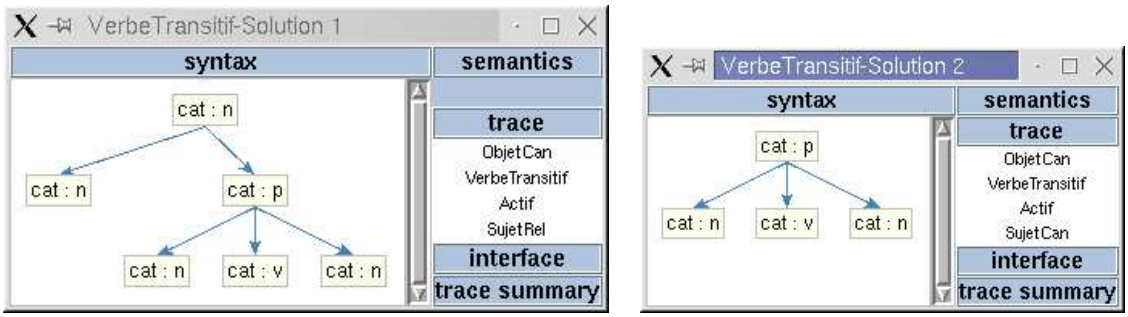


FIG. 1 – Arbres TAG produits par XMG

Les évaluations Une fois les classes de notre méta-grammaire écrites, nous pouvons demander à XMG de calculer tous les arbres TAG correspondants. Cela se fait en demandant l'*évaluation* de la classe : `value VerbeTransitif`

Résultat La classe VerbeTransitif génère deux solutions, qui correspondent aux arbres de la figure 1.

4 Intégration dans une chaîne de TALN

Le système XMG produit des grammaires au format XML[7] utilisables pour l'analyse syntaxique ou la génération :

Analyse syntaxique Nous avons vu en section 1 que l'un des buts de la production automatique de grammaires était d'éviter les problèmes liés à la redondance entre structures syntaxiques. Dans le cas de TAG, on évite aussi cette redondance en utilisant un lexique à 3 niveaux : lemmes, formes fléchies et structures syntaxiques. Dans ce contexte, la méta-grammaire produit non pas des arbres qui seraient associés chacun à une ou plusieurs unité(s) lexicale(s), mais un ensemble de structures syntaxiques *non-ancrées* L'association structure syntaxique – unité(s) lexicale(s) est alors réalisée par l'analyseur syntaxique. Pour cela, il est nécessaire d'ajouter à ces structures non-ancrées certaines informations sur la morphologie des unités lexicales qui peuvent être accueillies. Ces informations prennent la forme de structures de traits (voir (Crabbé, Gaiffe et Roussanaly, 2003)). XMG, par sa dimension *dyn* contenant des matrices attributs-valeurs, offre un support adéquat pour cette association.

Génération L'utilisation de grammaires produites automatiquement en génération pose des problèmes différents. Un générateur aura pour but de produire un ensemble de phrases à partir d'une représentation sémantique. Là aussi, pour éviter la redondance, il y a un découpage structures syntaxiques – unité(s) lexicale(s). Par contre, leur association s'accompagne de l'instanciation des arguments du prédicat sémantique dans l'arbre. Pour pouvoir accéder à ces arguments dans la structure syntaxique, nous pouvons utiliser la dimension *dyn* et la coindexation entre traits (voir (Gardent et Kow, 2004)).

[7] les DTD pour grammaires TAG et IG sont fournies dans le système XMG disponible librement à l'adresse `http://sourcesup.cru.fr/xmg`.

Conclusion

XMG offre un cadre de travail adapté au développement de grammaires de taille relativement importante (voir (Crabbé, 2005a)). Une grammaire TAG du Français à large couverture a ainsi pu être développée par B. Crabbé (Crabbé, 2005b), celle-ci est en cours d'évaluation en analyse syntaxique sur la suite de tests TSNLP. Les premiers résultats sont encourageants, la couverture de la grammaire produite étant supérieure à 75%. Pour donner un ordre d'idée, cette méta-grammaire contient 246 classes, représentant 55 familles (*ie* cadres de sous-catégorisation), et générant 5075 arbres non-ancrés en 20 minutes de compilation (sur Pentium 4 - 2,66 Ghz et 1 Go de mémoire vive).

Nous travaillons à l'heure actuelle à la production de grammaires à portée sémantique dans une optique d'analyse syntaxique combinée avec un calcul sémantique. Nous visons également l'intégration à XMG d'une bibliothèque de dimensions ayant chacune un langage de représentation propre. Cette bibliothèque a pour but d'offrir à l'utilisateur des outils adaptés lui permettant de créer des instances de méta-grammaire pour un formalisme cible arbitraire.

Références

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

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

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

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

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

Crabbé B. (2005b), Représentation informatique de grammaires fortement lexicalisées - Application à la grammaire d'arbres adjoints, *Thèse de Doctorat (á paraître), Université Nancy 2*.

Crabbé B., Gaiffe B., Roussanaly A. (2003), Une plate-forme de conception et d'exploitation d'une grammaire d'arbres adjoints lexicalisés, *Actes de la conférence TALN'2003 Batz-sur-mer*.

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

Duchier D., Niehren J. (2000), Dominance Constraints with Set Operators, *Proceedings of the First International Conference on Computational Logic (CL2000).*

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

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

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

Joshi A., Schabes Y. (1997), Tree-Adjoining Grammars , *Handbook of Formal Languages* , G. Rozenberg and A. Salomaa , Springer, Berlin, New York.

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