XMG : Un compilateur de métagrammaire multiformalisme et extensible

Denys Duchier Joseph Le Roux Yannick Parmentier

6 juin 2005

Plan

Grammaires lexicalisées (1)

Approche différente des **grammaires de règles**

- A chaque **mot** de la langue est associé un ensembles de structures décrivant les différents **usages**.

Grammaires lexicalisées (1)

Approche différente des **grammaires de règles**

- A chaque **mot** de la langue est associé un ensembles de structures décrivant les différents **usages**.
- Pas de généralisation

Grammaires lexicalisées (1)

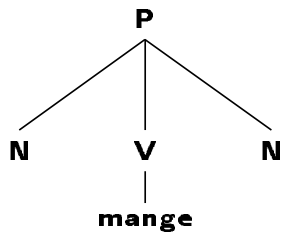
Approche différente des **grammaires de règles**

- A chaque **mot** de la langue est associé un ensembles de structures décrivant les différents **usages**.
- Pas de généralisation
- *Tout est exception !*

Grammaires lexicalisées (2)

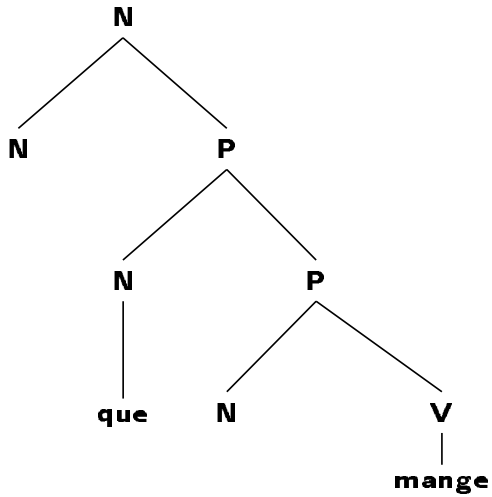
Exemple : « mange »

Jean mange la pomme



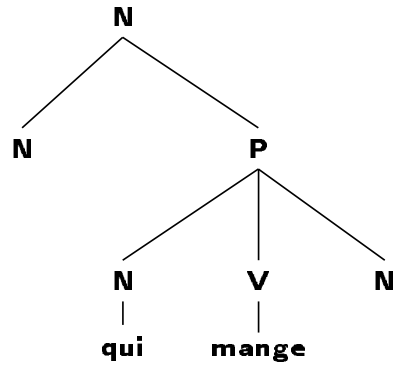
suj. can. et obj. can.

La pomme que Jean mange



suj. can. et obj. extr.

Jean qui mange une pomme



suj. extr. et obj. can.

Problèmes

Amplifiés par l'élargissement de la couverture :

Problèmes

Amplifiés par l'élargissement de la couverture :

- Pas assez d'*explication de la langue*

  (Que représente linguistiquement la branche  ?)

Problèmes

Amplifiés par l'élargissement de la couverture :

- Pas assez d'*explication de la langue*

- *Redondance structurelle*, rend difficile la maintenance et l'extension

  (la branche  est présente dans des milliers d'arbres)

Problèmes

Amplifiés par l'élargissement de la couverture :

- Pas assez d'*explication de la langue*
- *Redondance structurelle*, rend difficile la maintenance et l'extension
- *Ni modularisation ni partage* (la branche 

à modifier dans toutes les structures la contenant)

Solution proposée

Générer automatiquement la grammaire à partir d'une description concise et modulaire : **la métagrammaire**.

Solution proposée

Générer automatiquement la grammaire à partir d'une description concise et modulaire : **la métagrammaire**.

- Décrire des **fragments** de structures réutilisables qui ont un sens linguistique

Solution proposée

Générer automatiquement la grammaire à partir d'une description concise et modulaire : **la métagrammaire**.

- Décrire des **fragments** de structures réutilisables qui ont un sens linguistique
- Expliquer la formation des structures complètes par des **combinaisons** de ces fragments

Solution proposée

Générer automatiquement la grammaire à partir d'une description concise et modulaire : **la métagrammaire**.

- Décrire des **fragments** de structures réutilisables qui ont un sens linguistique
- Expliquer la formation des structures complètes par des **combinaisons** de ces fragments
- Garantir la **cohérence** de la grammaire durant ces combinaisons ou a posteriori

Principe (1)

L'entité de base est **la classe**.

$$Classe ::= Nom \rightarrow Definition \quad (1)$$

$$Definition ::= Description \mid Nom \mid Definition \wedge Definition \mid Definition \vee Definition \quad (2)$$

Principe (1)

L'entité de base est **la classe**.

$$Classe ::= Nom \rightarrow Definition \quad (1)$$

$$Definition ::= Description \mid Nom \mid Definition \wedge Definition \mid Definition \vee Definition \quad (2)$$

Exemple

$VerbeTransitif \rightarrow Sujet \wedge MorphologieVerbe \wedge Objet$
$Sujet \rightarrow SujetCanonique \vee SujetExtrait$
$Objet \rightarrow ObjetCanonique \vee ObjetExtrait$

Principe (2)

XMG et la programmation logique

XMG

Classe
Description
Accumulation
Alternative

Principe (2)

XMG et la programmation logique

XMG	Prog. Logique
Classe	Clause
Description	Fait
Accumulation	Conjonction
Alternative	Disjonction
	Requête

Principe (2)

XMG et la programmation logique

XMG	Prog. Logique
Classe	Clause
Description	Fait
Accumulation	Conjonction
Alternative	Disjonction
Evaluation	Requête

Principe (2)

XMG et la programmation logique

XMG	Prog. Logique
Classe	Clause
Description	Fait
Accumulation	Conjonction
Alternative	Disjonction
Evaluation	Requête

Métagrammaire $\equiv$ Programme logique
XMG $\equiv$ Compilateur + M.V. programmation logique

Contenu des descriptions

Différents types d'information : syntaxe/sémantique/lexique

- La sémantique n'est pas une surcouche.
- Formalismes indépendants
- Autres niveaux de la langue

Contenu des descriptions

Différents types d'information : syntaxe/sémantique/lexique

- La sémantique n'est pas une surcouche.
- Formalismes indépendants
- Autres niveaux de la langue

Idée : Les descriptions sont composées de plusieurs **dimensions**.

Exemple

TAG :

- *<**syn**> : Dimension syntaxique (descriptions d'arbre)*
- *<**sem**> : Dimension sémantique (ensembles de prédicats)*
- *<**dyn**> : Dimension interface lexique/syntaxe/sémantique (matrices traits-valeurs)*

Contenu des descriptions

Différents types d'information : syntaxe/sémantique/lexique

- La sémantique n'est pas une surcouche.
- Formalismes indépendants
- Autres niveaux de la langue

Idée : Les descriptions sont composées de plusieurs **dimensions**.

Exemple

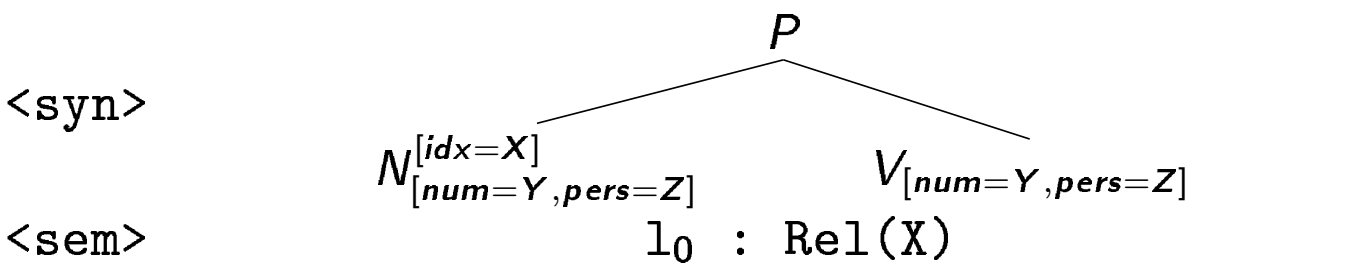
TAG :

- *<**syn**> : Dimension syntaxique (descriptions d'arbre)*
- *<**sem**> : Dimension sémantique (ensembles de prédicats)*
- *<**dyn**> : Dimension interface lexique/syntaxe/sémantique (matrices traits-valeurs)*

$Definition ::= < DIM > += Description \mid \ldots$

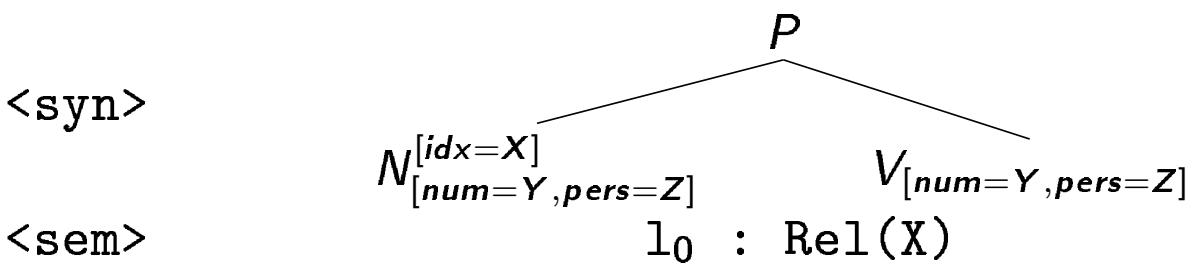
Syntaxe/Sémantique/Lexique (exemple)

On veut obtenir la structure `Intransitive` :



Syntaxe/Sémantique/Lexique (exemple)

On veut obtenir la structure `Intransitive` :



<dyn>
$$\begin{bmatrix} \text{head} & = & [\text{cat=v,sem=Rel}] \\ \text{arg1} & = & [\text{cat=n,fonct=suj,num=Y,pers=Z}] \end{bmatrix}$$

dors :
$$\begin{bmatrix} \text{head} & = & [\text{cat=v,sem=dormir}] \\ \text{arg1} & = & [\text{cat=n,fonct=suj,num=sg,pers=1|2}] \end{bmatrix}$$

`Intransitive` = Sujet $\wedge$ MorphV $\wedge$ PUnaire $\wedge$ ...

Syntaxe/Sémantique/Lexique (exemple)(2)

$$\begin{array}{lll} \text{MorphV} \rightarrow & \texttt{<syn>+=} & \begin{array}{c} P \\ | \\ V \end{array} \\ \wedge & \texttt{<dyn>+=} & [\text{head} = [\text{cat=v,sem=R}] \end{array}$$

Syntaxe/Sémantique/Lexique (exemple)(2)

$$\begin{array}{lcll} \text{MorphV} & \rightarrow & \texttt{<syn>+=} & \begin{array}{c} P \\ | \\ V \end{array} \\ & \wedge & \texttt{<dyn>+=} & [\text{head} = [\text{cat=v,sem=R}] \end{array}$$

$$\begin{array}{lcll} \text{PUnaire} & \rightarrow & \texttt{<sem>+=} & l_0 : \texttt{Rel}(I_1) \\ & \wedge & \texttt{<dyn>+=} & [\text{head} = [\text{sem=Rel}]] \end{array}$$

Syntaxe/Sémantique/Lexique (exemple)(2)

MorphV $\rightarrow$ <syn>+= P / V

$\wedge$ <dyn>+= [head = [cat=v,sem=R]

PUnaire $\rightarrow$ <sem>+= l_0 : Rel(I_1)

$\wedge$ <dyn>+= [head = [sem=Rel]]

Sujet $\rightarrow$ <syn>+= P / $N_{[num=Y,pers=Z]}^{[idx=I]}$ $V_{[num=Y,pers=Z]}$

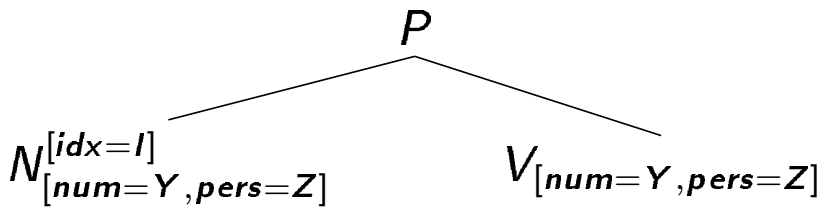
$\wedge$ <dyn>+= [arg1 = [cat=n,fonct=suj,num=Y,pers=Z]]

Syntaxe/Sémantique/Lexique (exemple)(2)

MorphV → <syn>+= 

∧ <dyn>+= [head = [cat=v,sem=R]

PUnaire → <sem>+= l_0 : Rel(I_1)
∧ <dyn>+= [head = [sem=Rel]]

Sujet → <syn>+= 

∧ <dyn>+= [arg1 = [cat=n,fonct=suj,num=Y,pers=Z]]

... ≡ MorphV.P = Sujet.P ∧ MorphV.V = Sujet.V
∧ Sujet.I=PUnaire.I_1

Multiformalisme et extensibilité

Dimensions : architecture souple pour évoluer rapidement

Multiformalisme et extensibilité

Dimensions : architecture souple pour évoluer rapidement

Nouvelle dimension :

1. Nouveau langage de description
2. Opération d'accumulation (unification)

Pour l'instant : *en dur*
A l'avenir : paramétrable par les utilisateurs

Accumulation et espaces de nom

La **portée** des variables : la classe

Accumulation et espaces de nom

La **portée** des variables : la classe

Comment accéder aux variables d'une classe accumulée ?

- par défaut : notation pointée (*cf* exemple précédent)
- import/export d'espace de nom.

Exemple

```
class A ... export X,Y,Z ...
class B ... import A
class C ... import A as [X=U,...]
```

Contraintes additionnelles

Exprimer des contraintes

Exemple

$VerbeTransitif \rightarrow Sujet \wedge MorphologieVerbe \wedge Objet$
$Sujet \rightarrow SujetCanonique \vee SujetExtrait$
$Objet \rightarrow ObjetCanonique \vee ObjetExtrait$

On veut éviter Sujet extrait et Objet extrait.

2 Solutions :

Contraintes additionnelles

Exprimer des contraintes

Exemple

$VerbeTransitif \rightarrow Sujet \wedge MorphologieVerbe \wedge Objet$
$Sujet \rightarrow SujetCanonique \vee SujetExtrait$
$Objet \rightarrow ObjetCanonique \vee ObjetExtrait$

On veut éviter Sujet extrait et Objet extrait.

2 Solutions :

- Classes en exclusion mutuelle

Contraintes additionnelles

Exprimer des contraintes

Exemple

$VerbeTransitif \rightarrow Sujet \wedge MorphologieVerbe \wedge Objet$
$Sujet \rightarrow SujetCanonique \vee SujetExtrait$
$Objet \rightarrow ObjetCanonique \vee ObjetExtrait$

On veut éviter Sujet extrait et Objet extrait.

2 Solutions :

- Classes en exclusion mutuelle
- Résolveur de contraintes
  - filtrage de structures
  - unification supplémentaires implicites

Contraintes additionnelles

Exprimer des contraintes

Exemple

$VerbeTransitif \rightarrow Sujet \wedge MorphologieVerbe \wedge Objet$
$Sujet \rightarrow SujetCanonique \vee SujetExtrait$
$Objet \rightarrow ObjetCanonique \vee ObjetExtrait$

On veut éviter Sujet extrait et Objet extrait.

2 Solutions :

- Classes en exclusion mutuelle
- Résolveur de contraintes
  - filtrage de structures
  - unification supplémentaires implicites

Contraintes additionnelles

Exprimer des contraintes

Exemple

$VerbeTransitif \rightarrow Sujet \wedge MorphologieVerbe \wedge Objet$
$Sujet \rightarrow SujetCanonique \vee SujetExtrait$
$Objet \rightarrow ObjetCanonique \vee ObjetExtrait$

On veut éviter Sujet extrait et Objet extrait.

2 Solutions :

- Classes en exclusion mutuelle
- Résolveur de contraintes
  - filtrage de structures
  - unification supplémentaires implicites

  Exemple : sensibilité aux ressources
  - une extraction max. par arbre
  - Nœuds avec polarités

Utilisation

Schéma d'utilisation :

1. Ecriture de la métagrammaire (programme logique)
2. Compilation
3. Execution
4. Résolution de contraintes additionnelles

Résultat : structures grammaticales classées par famille

- XML (TAGML)
- Affichage à l'écran

Quelques résultats

Utilisé au LORIA :

- (Crabbé) **TAG** 246 classes pour 55 familles : 5075 arbres non ancrés. (5 min. compilation+exécution et 15 min. résolution)
- (Gardent) **TAG** Une grammaire orientée sémantique et génération
- (Perrier) **GI** 237 classes pour 45 familles : 322 descriptions d'arbre. Pas de résolution de contraintes (compilation+exécution : 1 min).

Conclusion

XMG : Cadre général multiformalisme + logiciel

1. - Langage de description de structures complètement déclaratif
   - Dimensions : plusieurs infos, plusieurs formalismes
   - Extensible : dimensions et contraintes

2. - Programmation logique (techniques efficaces) appliquée à la construction de grammaires
   - Compilateur + Machine virtuelle (portabilité) (Mozart/Oz)
   - Résolution de contraintes

Disponible (licence CeCill) sur `http ://sourcesup.cru.fr/xmg/`